Introduction to Pointers

Unit 5 Objective

- Understand what a pointer variable is
- Declare and initialize pointers
- Use pointers to traverse arrays
- Pass pointers to functions
- Write and use functions that return pointers
- Access command line arguments

Pointers Overview

- What is a pointer?

  A variable that contains an address

- Why use pointers?

  To efficiently access data

  To write flexible code

  To change variables passed to a function

  To work with dynamically allocated memory

  To access hard-coded addresses in system code

Declaring, Initializing, and Using Pointers

```
#include  <stdio.h>
main()
{
   int  num1=3, num2=6 ;
   int  *p ;

   p = &num1 ;
   printf("%d\n", *p) ;

   *p = 20 ;
   printf("%d", *p) ;
   printf("%d\n", num1) ;

   p = &num2 ;
   printf("%d\n", *p) ;
}
```

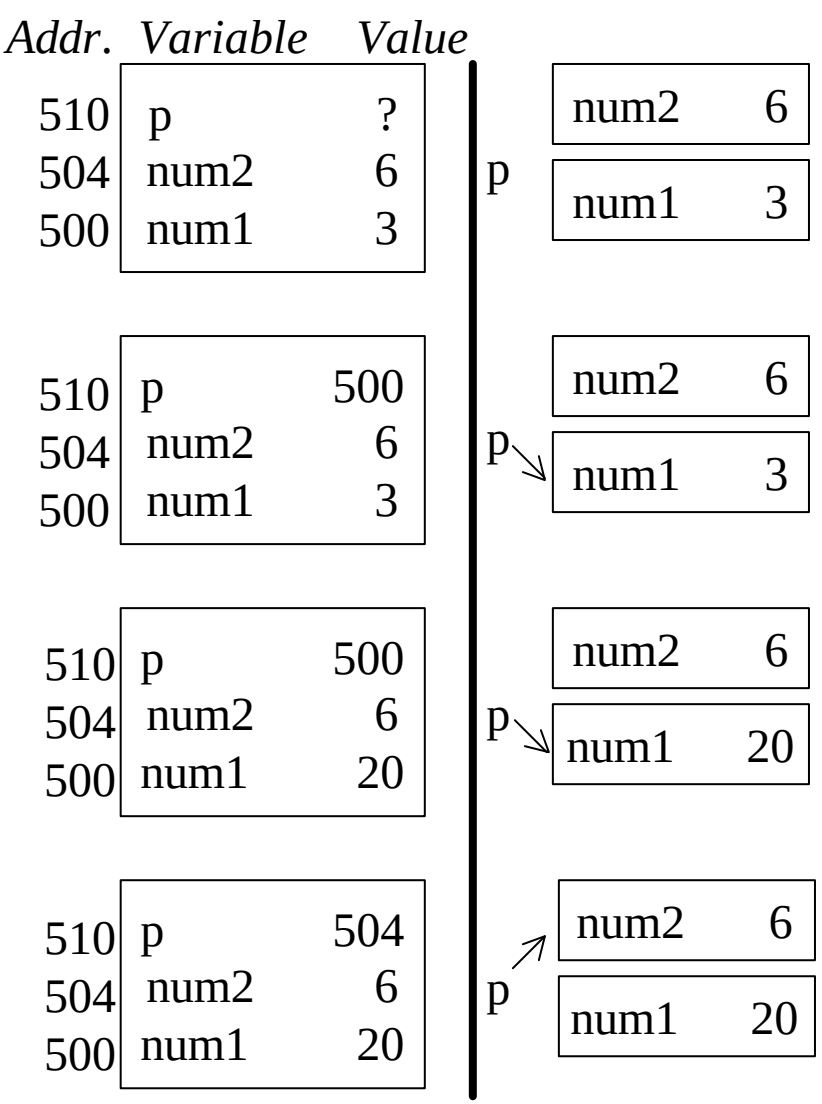
Output:

3

20 20

6

Hypothetical Stack



Pointer Exercise - & and *

1. What is the output of the following program?

```
 1   #include    <stdio.h>
 2
 3   main()
 4   {
 5         int         count = 10, x;
 6         int         *ip;
 7
 8         ip = &count;
 9         x = *ip;
10
11         printf("count = %d, x = %d\n", count, x);
12   }
```

2. What is the output of the following program?

```
 1   #include    <stdio.h>
 2
 3   main()
 4   {
 5         int         i1, i2;
 6         int         *p, *q;
 7
 8         i1 = 5;
 9         p = &i1;
10         i2 = *p / 2 + 10;
11         q = p;
12         printf("i1 = %d, i2 = %d, ", i1, i2);
13         printf("*p = %d, *q = %d\n", *p, *q);
14   }
```

(Continued on next page)

Pointer Exercise - & and *, continued

3. What is wrong with this program?

```
1   #include        <stdio.h>
2
3   main()
4   {
5           int         num1 = 100, *p ;
6
7           printf("num1 is %d\n", num1) ;
8           printf("*p is %d\n", *p) ;
9   }
```

   Will it compile?

   Will it run?

   What will be printed?

4. Suppose that i is an integer and p is an integer pointer. What does the following phrase mean?

   p "points to" i

Pointer Arithmetic

- Important to declare pointer with correct type

- OK to add to, subtract from, compare pointers and subtract one pointer from another

char *p;

*p is a character

p++ is equivalent to p = p + (1 byte)

int *p;

*p is an integer

p++ is equivalent to p = p + (bytes in an int)

Using a Pointer to Traverse an Array

- For sequential access, * is faster than [ ]

```
 1  /* Reads an input line into an array */
 2  /* Assumes input line is <= LINELEN chars */
 3
 4  #include     <stdio.h>
 5  #define      LINELEN     256
 6
 7  main()
 8  {
 9      char     line[LINELEN + 1], *p;
10
11      p = line;      /* p = &line[0]; */
12      while ((*p = getchar()) != '\n')
13          p++;
14      *p = '\0';
15      printf("%s\n",line);
16  }
```

```
$ a.out
do it    <-- typed by user
do it    <-- output from program
$
```

Passing an Address to a Function

- Allows a function to change the variable whose address is passed
- The function parameter is declared to be a pointer

```
 1  #include   <stdio.h>
 2
 3  main()
 4  {
 5      int    num1 = 10, num2 = 5;
 6      void swap();
 7
 8      printf("%d %d\n", num1, num2);
 9      swap(&num1, &num2);
10      printf("%d %d\n", num1, num2);
11  }
12
13  void swap(one, two)
14  int *one, *two;
15  {
16      int    temp;
17
18      temp = *one;
19      *one = *two;
20      *two = temp;
21  }
```

Output:

```
10 5
5 10
```

Passing an Array Address to a Function

```
 1  /* Shows 2 ways of declaring a function */
 2  /* parameter when an array address is passed */
 3  main()
 4  {
 5      char    str1[20], str2[20];
 6      void    array_cpy(), ptr_cpy();
 7
 8      ...
 9      array_cpy(str1, str2);  /* ptr_cpy(str1, str2)
10      ...
11  }
12
13  void array_cpy(one, two)
14  char   one[], two[];
15  {
16      int     i;
17      for (i = 0; two[i] != '\0'; i++)
18              one[i] = two[i];
19      one[i] = '\0';
20  }
21
22
23  void ptr_cpy(one, two)
24  char   *one, *two;
25  {
26      for ( ;*two != '\0'; one++, two++ )
27              *one = *two;
28      *one = '\0';
29  }
```

How a Function Returns a Pointer

```
 1      /* main() passes the address of an array */
 2      /* holding city and state to getstate()  */
 3      /* which returns a pointer to the state. */,
 4      #include    <stdio.h>
 5
 6      main()
 7      {
 8  ☞       char    *getstate();
 9          static char   info[] = "Palmyra, NJ";
10
11          printf("State is %s\n", getstate(info))
12      }
13
14      /* Assumes p points to string */
15      /* in form "City, State" */
16
17  ☞   char *getstate(p)
18      char    *p;
19      {
20          while ( *p != ',')
21                  p++;
22          return(p + 2);
23      }
```

Output:

```
State is NJ
```

The Null Pointer

- **It is not legal to read from or write to address 0**

- **A null pointer:**

  Pointer with value of 0

  Returned by pointer-returning functions to signify failure

- **NULL is defined in stdio.h as**

  #define NULL 0

  -or-

  #define NULL (void*) 0

Get String Function - gets( )
Put String Function - puts( )

SYNOPSIS

```
#include <stdio.h>
char *gets(s)
char *s;
```

DESCRIPTION

gets reads one line from *stdin* into the array pointed to by s. The newline character is discarded. The string is null-terminated. Returns NULL on end-of-file if no characters read.

SYNOPSIS

```
#include <stdio.h>
int puts(s)
char *s
```

DESCRIPTION

puts writes the null-terminated string pointed to by *s*, followed by a new-line character to *stdout*.

gets() and puts() - Example

```
1   #include        <stdio.h>
2   #define         LINELEN         256
3   main()
4   {
5           char    name[LINELEN + 1] ;
6
7           printf("Enter full name: ") ;
8           if (gets(name) == (char *)NULL)
9                   printf("Error\n") ;
10          else
11                  puts(name) ;
12  }
```

Output:

Enter full name: Jean M. Griffin

Jean M. Griffin

Some String-Handling Functions

SYNOPSIS

```
#include <string.h>

char *strcat(s1, s2)
char *s1, *s2;

int strcmp(s1, s2)
char *s1, *s2;

char *strcpy(s1, s2)
char *s1, *s2;

int strlen(s)
char *s;
```

DESCRIPTION

strcat appends s2 to s1 and returns s1.

strcmp's return value is less than, equal to, or greater than 0 if s1 is lexicographically less than, equal to, or greater than s2.

strcpy copies s2, to s1, stopping after the null is copied, s1 is returned.

strlen returns the number of characters up to but not including the '\0' character.

String Functions - Examples

```
 1  #include    <string.h>
 2  #include    <stdio.h>
 3  #define     LINELEN     256
 4  #define     MINLEN      6
 5  #define     MAXLEN      12
 6  char        input[LINELEN + 1], passwd[MAXLEN +1];
 7
 8  change_passwd()
 9  {
10      printf("Enter new password: ");
11      gets(input);
12      if (strlen(input) < MINLEN) {
13              printf("Password too short.\n");
14              exit(1);
15              }
16      ...
17  }
18
19  verify_passwd()
20  {
21      printf("Enter password: ");
22      gets(input);
23      getpassword(passwd);  /* local function that */
24          /*copies current password to supplied addr.*/
25      if (strcmp(input,passwd) != 0) {
26              printf("Sorry\n");
27              exit(2);
28              }
29      ...
30  }
```

Converting an ASCII String to Integer, Long, or Double

SYNOPSIS

```
int  atoi(str)
char *str;        /* char str[]; */

long  atol(str)
char *str;        /* char str[]; */

double  atof(str)
char *str;        /* char str[]; */
```

EXAMPLES

```
 1  #include <stdio.h>
 2
 3  main()
 4  {
 5      int     quantity;
 6      double  percentage, atof();
 7      char    line[81];
 8      ...
 9      scanf("%80s", line);
10      quantity = atoi(line);
11      ...
12      scanf("%80s", line);
13      percentage = atof(line);
14      ...
15  }
```

Common Pointer Idioms

```
/* Returns the number of */
/* characters up to but not including */
/* the '\0' character */
int strlen(s)
register char *s;
{
register char *p;

    p = s;
```

`while(*p != '\0')`
`  p++;`
`return(p-s);`
	`while (*p)`
	`  p++;`
	`return(p-s);`
		`while(*p++)`
		`  ;`
		`return((p-s) - 1);`

```
}
```

Accessing Command Line Arguments

- argc - argument count
- argv - argument vector

```
 1  /* Prints words from command line */
 2  #include    <stdio.h>
 3
 4  main(argc, argv)
 5  int      argc;
 6  char     *argv[];
 7  {
 8      int     i;
 9
10      printf("Number of words: %d\n", argc);
11
12      for ( i = 0; i < argc; i++ )
13          printf("%s\n", argv[i]);
14  }
```

```
   argv[0]    argv[1]   argv[2]
$ a.out      file1     file2
Number of words: 3
a.out
file1
file2
```