C-Programming Practice

Part – 3

Pointers

Suppose v is a variable that represents some particular data item. The compiler will automatically assign

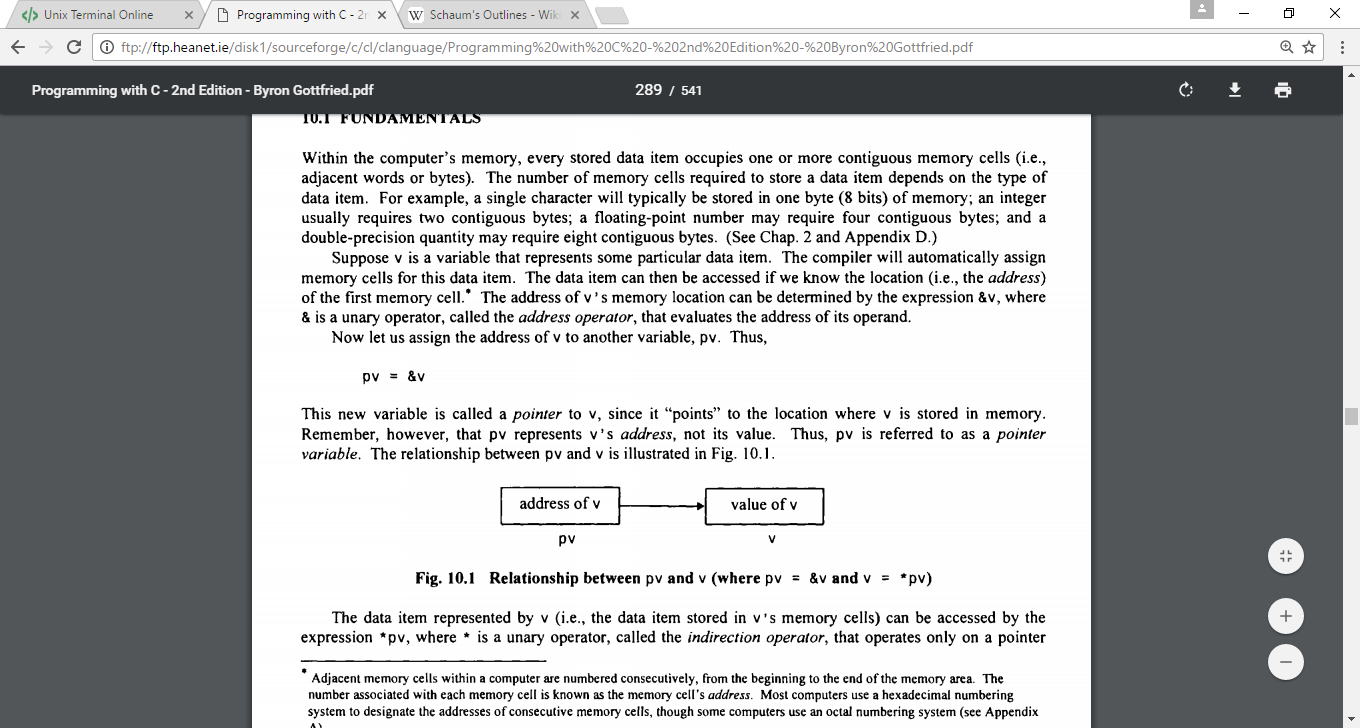
memory cells for this data item. The data item can then be accessed if we know the location (i.e., the address) of the first memory cell. The address of **v** ’s memory location can be determined by the expression **&v**, where **&** is a unary operator, called the address operator, that evaluates the address of its operand. Now let us assign the address of **v** to another variable, **pv**. Thus,

**pv = &v**

This new variable is called a pointer to **v**, since it “points” to the location where v is stored in memory.

Remember, however, that **pv** represents **v**’s address, not its value. Thus, **pv** is referred to as a pointer

variable. The relationship between **pv** and **v** is illustrated in the following figure:



The data item represented by **v** (i.e., the data item stored in v’s memory cells) can be accessed by the expression **\*pv**, where \* is a unary operator, called the indirection operator, that operates only on a pointer. If we write **pv = &v** and **U = \*pv**, then **U** and **v** will both represent the same value; i.e., the value of **v** will indirectly be assigned to **U**. (It is assumed that **U** and **v** are of the same data type.)

Examples:

#include <stdio.h>

int main(void) {

    int u = 3;

    int v;

    int \*pu; /\* pointer to an integer \*/

    int \*pv; /\* pointer to an integer \*/

    pu = &u; /\* assign address of U to pu \*/

    v = \*pu; /\* assign value of U to v \*/

    pv = &v; /\* assign address of v to pv \*/

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

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

    return 0;

}

$ gcc pointers1.c -o pointers1.out

$ ./pointers1.out

u=3 &u=0x7ffed54c27c0 pu=0x7ffed54c27c0 \*pu=3

v=3 &v=0x7ffed54c27c4 pv=0x7ffed54c27c4 \*pv=3

$

|  |  |
| --- | --- |
| #include <stdio.h>  int main() {  int v = 3;  **int \*pv;**  pv = &v; /\* pv points to v \*/  printf ("\n\*pv=%d v=%d", \*pv, v) ;  \*pv = 0; /\* reset v indirectly \*/  printf("\n\*pv=%d v=%d\n", \*pv, v);  return 0;  } | #include <stdio.h>  int main(void) {  int u1, u2;  int v = 3;  **int \*pv;** /\* pv points to v \*/  **u1 = 2 \* (v + 5);**  pv = &v;  **u2 = 2 \* (\*pv + 5);**  printf("\nu1=%d u2=%d\n", u1, u2);  return 0;  } |

Pointers and Functions

Example: Observe how pointers can be used to change the values of variables used in the function call. **This is one mechanism used for returning more than one value from a function.**

Compile and run the following example to observe this behavior:

#include <stdio.h>

void functl(int, int); /\* function prototype \*/

void funct2(int\*, int\*); /\* function prototype \*/

int main() {

int u = 1;

int v = 3;

printf("\nBefore calling funct1: u=%d v=%d", u, v);

functl(u, v);

printf("\nAfter calling funct1: u=%d v=%d\n", u, v);

printf("\nBefore calling funct2: u=%d v=%d", u, v) ;

funct2(&u, &v);

printf("\nAfter calling funct2: u=%d v=%d\n", u, v);

return 0;

}

void functl(int u, int v) {

u = 0; v = 0;

printf( "\nWithin funct1: u=%d v=%d", u, v);

return;

}

void funct2(int \*pu, int \*pv) {

\*pu = 0;

\*pv = 0;

printf("\nWithin funct2: \*pu=%d \*pv=%d" , \*pu , \*pv) ;

return ;

}

$ gcc pointers4.c -o pointers4.out

$ ./pointers4.out

Before calling funct1: u=1 v=3

Within funct1: u=0 v=0

After calling funct1: u=1 v=3

Before calling funct2: u=1 v=3

Within funct2: \*pu=0 \*pv=0

After calling funct2: u=0 v=0

$

Example: Using Functions With Pointers for analyzing a line of text

#include <stdio.h>

#include <ctype.h>

void scanline(char line[], int \*pv, int \*pc, int \*pd, int \*pw, int \*po) {

/\* analyze the characters in a line of text. \*/

char c; /\* uppercase character \*/

int count = 0; /\* character counter \*/

printf("You entered %s\n", line);

while ((c = toupper(line[count])) != '\0') {

if(c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') ++\*pv;

else if (c >= 'A' && c <= 'Z') ++\*pc; /\* consonant \*I

else if (c >= '0' && c <= '9') ++\*pd; /\* digit \*/

else if (c == ' ' || c == '\t') ++\*pw; /\* whitespace \*/

else ++\*po; /\* other \*/

++count;

}

return;

}

int main()

{

char line[80]; /\* line of text \*/

int vowels = 0; /\* vowel counter \*/

int consonants = 0; /\* consonant counter \*/

int digits = 0; /\* digit counter \*/

int whitespc = 0; /\* whitespace counter \*/

int other = 0; /\* remaining character counter \*/

printf("Enter a line of text below:\n");

fgets(line, 80, stdin); //SAFER ALTERNATIVE TO gets()

scanline(line, &vowels, &consonants, &digits, &whitespc, &other);

printf("\nNo. of vowels: %d", vowels);

printf("\nNo. of consonants: %d", consonants);

printf ("\nNo. of digits: %d", digits);

printf("\nNo. of whitespace characters: %d", whitespc);

printf("\nNo. of other characters: %d\n", other);

return 0;

}

**Exercise 1: Write a function called swap4 that takes as input pointers to two integer and two float variables i.e. swap(int\*, int\*, float\*, float\*). It then returns swapped values of the integer and float variables. Write a main function to test your program.**

**Exercise 2: Write a function that receives time in seconds and returns the equivalent time in hours, minutes, and seconds. Write a main function to test your function. For example if the received time is 4000 seconds, the function returns 1 hour, 6 minutes, and 40 seconds.**

**Points to Ponder (Pointers):**

🡆 *After studying pointers and functions, what can you conclude about the ‘&’ in scanf? Is it a pointer to a variable?*

Pointers and Arrays (1-D)

An array name is really a pointer to the first element in the array. Therefore, if x is a one-dimensional array, then the address of the first array element can be expressed as either **&x[0]** or simply as **x**. In general, the address of array element **(i + 1)** can be expressed as either **&x[i]** or as **(x + i).**

Example:

#include <stdio.h>

int main()

{

    static int x[10] = {10, 11, 12, 13, 14, 15, 16, 17, 18, 19};

    int i;

    for (i = 0; i <= 9; i++) {

        /\* display an array element \*/

        printf("i = %d\tx[i] = %d\t\*(x+i) = %d\t\n", i, x[i], \*(x+i));

        /\* display the corresponding array address \*/

        printf ("&x[i] = %p\tx+i = %p\n", &x[i], (x+i));

    }

    return 0;

}

Note: In ‘C’ a char array (a string) can also be declared as char\* (a pointer to a char). This is because a pointer to a char is simply the address of the first location of a memory address containing a char.

Example:

#include <stdio.h>

#include <stdlib.h>

int main()

{

char \*s;

s = (char \*) malloc (80 \* sizeof(char));

//allocates memory of 80 chars to s. Similar to char s[80]

gets(s); //ignore the warning for this example

puts(s);

return 0;

}

**Exercise 3: Write a C program that will enter a line of text, store it in an array and then display it backwards. Write a function void reverse(char \*line) that returns the reversed line of text. Use the fact that strings end in ‘\0’.**

**Points to Ponder (Pointers):**

🡆 *After studying pointers and 1-D arrays, why do we write*

char name[80]; scanf(“%s”, name); [and not scanf(“%s”, &name);]

# Structures and Pointers

Consider the following example:

#include <stdio.h>

struct person

{

   int age;

   float weight;

};

int main()

{

    struct person person1;

    struct person \*personPtr;

    personPtr = &person1;    // Referencing pointer to memory address of person1

    printf("Enter age: ");

    scanf("%d",&(\*personPtr).age);

    printf("Enter weight: ");

    scanf("%f",&(\*personPtr).weight);

    printf("Displaying: ");

    printf("%d, %f\n",(\*personPtr).age,(\*personPtr).weight);

    printf("The same can be done using the -> notation\n");

    printf("%d, %f\n", personPtr->age, personPtr->weight);

    return 0;

}

**Exercise 4: Consider the following code for a Linked List in C. Compile and execute this program and study various method. Write a function struct node\* findLast() that finds and returns the last element of the linked list. Print this struct in the format (key, data) in main.**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct node {

   int data;

   int key;

   struct node \*next;

};

struct node \*head = NULL; //global variable

struct node \*current = NULL; //global variable

void printList() { //display the list

   struct node \*ptr = head;

   printf("\n[ ");

   //start from the beginning

   while(ptr != NULL) {

      printf("(%d, %d) ",ptr->key,ptr->data);

      ptr = ptr->next;

   }

   printf(" ]");

}

//insert link at the first location

void insertFirst(int key, int data) {

   //create a link

   struct node \*link = (struct node\*) malloc(sizeof(struct node));

   link->key = key;

   link->data = data;

   //point it to old first node

   link->next = head;

   //point first to new first node

   head = link;

}

//delete first item

struct node\* deleteFirst() {

   //save reference to first link

   struct node \*tempLink = head;

   //mark next to first link as first

   head = head->next;

   //return the deleted link

   return tempLink;

}

//is list empty

int isEmpty() {

   return head == NULL;

}

int length() {

   int length = 0;

   struct node \*current;

   for(current = head; current != NULL; current = current->next) {

      length++;

   }

   return length;

}

//find a link with given key

struct node\* find(int key) {

   //start from the first link

   struct node\* current = head;

   //if list is empty

   if(head == NULL) {

      return NULL;

   }

   //navigate through list

   while(current->key != key) {

      //if it is last node

      if(current->next == NULL) {

         return NULL;

      } else {

         //go to next link

         current = current->next;

      }

   }

   //if data found, return the current Link

   return current;

}

//delete a link with given key

struct node\* delete(int key) {

   //start from the first link

   struct node\* current = head;

   struct node\* previous = NULL;

   //if list is empty

   if(head == NULL) {

      return NULL;

   }

   //navigate through list

   while(current->key != key) {

      //if it is last node

      if(current->next == NULL) {

         return NULL;

      } else {

         //store reference to current link

         previous = current;

         //move to next link

         current = current->next;

      }

   }

   //found a match, update the link

   if(current == head) {

      //change first to point to next link

      head = head->next;

   } else {

      //bypass the current link

      previous->next = current->next;

   }

   return current;

}

int main(void) {

   insertFirst(1, 10);

   insertFirst(2, 20);

   insertFirst(3, 30);

   insertFirst(4, 1);

   insertFirst(5, 40);

   insertFirst(6, 56);

   printf("Original List: ");

   //print list

   printList();

   while(!isEmpty()) {

      struct node \*temp = deleteFirst();

      printf("\nDeleted value:");

      printf("(%d, %d) ",temp->key,temp->data);

   }

   printf("\nList after deleting all items: ");

   printList();

   insertFirst(1, 10);

   insertFirst(2, 20);

   insertFirst(3, 30);

   insertFirst(4, 1);

   insertFirst(5, 40);

   insertFirst(6, 56);

   printf("\nRestored List: ");

   printList();

   printf("\n");

   struct node \*foundLink = find(4);

   if(foundLink != NULL) {

      printf("Element found: ");

      printf("(%d,%d) ",foundLink->key,foundLink->data);

      printf("\n");

   } else {

      printf("Element not found.");

   }

   delete(4);

   printf("List after deleting an item: ");

   printList();

   printf("\n");

   foundLink = find(4);

   if(foundLink != NULL) {

      printf("Element found: ");

      printf("(%d, %d) \n",foundLink->key,foundLink->data);

      printf("\n");

   } else {

      printf("Element not found.\n");

   }

   return 0;

}

# Text Files

**Example 1**: The program below reads *miles* from **data.txt**, displays the value on screen, and writes the corresponding *kilometers* to **result.txt**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #define KMS\_PER\_MILE 1.609  int main(void) {  double kms, miles;  FILE \*infile, \*outfile;  infile = fopen("data.txt","r");  if(infile == NULL){  printf("Error: Failed to open data.txt\n");  exit(1);  }  outfile = fopen("result.txt","w");  fscanf(infile, "%lf", &miles);  fprintf(outfile, "The distance in miles is %.2f.\n", miles);  kms = KMS\_PER\_MILE \* miles;  fprintf(outfile, "That equals %.2f kilometers.\n", kms);  fclose(infile);  fclose(outfile);  return 0;  } |

Instead of using fcanf and printf we can use fgetc and fputc as shown below:

#include <stdio.h>

#include <stdlib.h>

int main (void){

FILE \*in;

in = fopen("example2.c","r");

if(in == NULL){

printf("Error: Failed to open example2.c\n");

system("PAUSE");

exit(1);

}

char ch;

while((ch = fgetc(in)) != EOF)

fputc(ch, stdout);

fclose(in);

return 0;

}

The following program requires an introduction to

int main(int argc, char \*argv[])

Explore main functions in C of this format. What does argc and argv actually mean?

**Exercise 5:** Write a utility **mycat** in ‘C’ that emulates the functionality of the linux ‘**cat’** command, i.e., displays the contents of a text file. Once you are done with the basic functionality, try to (a) increase the functionality by incorporating the ‘>’ symbol for writing to files and the ‘>>’ symbol for appending to files.

Some of the basic modes for opening files in ‘C’ are:

The basic modes for opening files are:

|  |  |
| --- | --- |
| "r" | Open a text file for reading. Error if the file does not exist |
| "r+" | Open a text file for reading and writing. Error if the file does not exist |
| "w" | Open a text file for writing and create the file if it does not exist. If the file exists then make it blank. |
| "w+" | Open a text file for reading and writing and create the file if it does not exist. If the file exists then make it blank. |
| "a" | Open a text file for appending (writing at the end of file) and create the file if it does not exist. |
| "a+" | Open a text file for reading and appending and create the file if it does not exist. |