Chapter 11 Pointers

- ในบทนี้เราจะได้เรียนรู้เกี่ยวกับ pointers ซึ่งเป็นเรื่องที่สำคัญในภาษา C
- Pointers เป็นสิ่งหนึ่งที่ทำให้ภาษา C มีประสิทธิภาพและความยืดหยุ่น
- Pointers ทำให้สามารถจัดการโครงสร้างของข้อมูลที่ซับซ้อน, เปลี่ยนข้อมูลที่ส่งเป็น argument เข้าไปในฟังก์ชัน, จัดการหน่วยความจำแบบ dynamics, และจัดการ array ได้อย่างมีประสิทธิภาพ

9.1 การประกาศตัวแปร pointer

- สมมติว่ามีตัวแปรชื่อ count ถูกประกาศเป็นชนิด integer ดังตัวอย่าง

  int count = 10;
- เราสามารถประกาศตัวแปร pointer ขึ้นมาตัวหนึ่ง เพื่อใช้เข้าถึงข้อมูลในตัวแปร count ได้

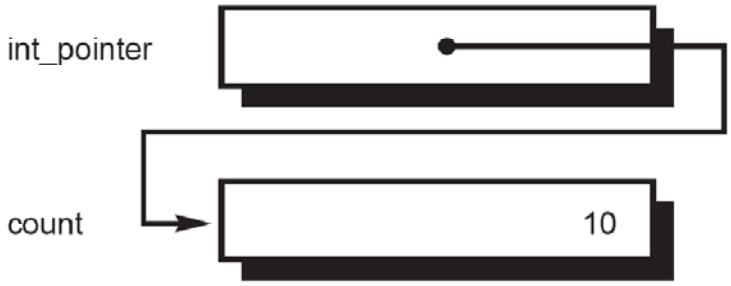
  int *int_pointer;

  โดยที่ดอกจันทร์นำหน้าชื่อตัวแปร ใช้กำหนดว่าตัวแปร int_pointer เป็น pointer สำหรับตัวแปร integer
- ก่อนที่เราจะนำ pointer ไปใช้งานได้ จะต้องมีการใส่ค่า address หรือที่อยู่ของตัวแปรในหน่วยความจำ ลงใน pointer ก่อน
- ในการนี้จะใช้ & หรือ address operator (เราได้ใช้ & ในคำสั่ง scanf มาก่อนแล้ว) วางหน้าชื่อตัวแปรใด ๆ เพื่อให้ตัวแปรแสดง address ของมันออกมา

(เมื่อมีการประกาศตัวแปรใด ๆ จะมีการจองพื้นที่ในหน่วยความจำไว้ส่วนหนึ่งเพื่อเก็บข้อมูลของตัวแปรนั้น ดังนั้นแต่ละตัวแปรจึงมี address หรือที่อยู่ในหน่วยความจำ)

- เราได้ประกาศตัวแปรชื่อ count และ pointer ชื่อ int_pointer เอาไว้แล้ว ดังนั้นการใส่ address ของ count ลงใน int_pointer ทำได้โดยใช้คำสั่ง

  int_pointer = &count;
- รูปแสดงความสัมพันธ์ระหว่าง int_pointer และ count



- เพื่อที่จะเข้าถึงข้อมูลที่อยู่ใน count (ในที่นี้คือเลข 10) ผ่านทางตัวแปร int_pointer ซึ่งเก็บ address ของ count อยู่เราจะใช้ indirection operator '*'

  x = *int_pointer;

  ซึ่งจะหมายถึงการนำข้อมูลที่ถูกชี้โดยตัวแปร int_pointer ไปใส่ใน x

Program 11.1 Illustrating Pointers

```
// Program to illustrate pointers
#include <stdio.h>
int main (void)
{
    int count = 10, x;
    int *int_pointer;
    int_pointer = &count;
    x = *int_pointer;
    printf("count = %i, x = %i\n", count, x);
    return 0;
}
```

```
count = 10, x = 10
```

- จากโปรแกรม ตัวแปร count และ x ถูกประกาศให้เป็นตัวแปร integer ตามปกติ
- บรรทัดถัดมา ตัวแปร int_pointer ประกาศให้เป็นตัวแปร pointer ชนิด integer
- เราสามารถรวมสองบรรทัดเข้าด้วยกันได้เป็น

  int count = 10, x, *int_pointer;
- หลังจากประกาศตัวแปร เราใช้ address operator เพื่อเอาค่า address ของตัวแปร count ไปเก็บใน int_pointer
- เมื่อ address ถูกเก็บลงใน int_pointer แล้วเราสามารถนำตัวแปรนี้ไปเข้าถึงข้อมูลของ count ได้โดยใช้ indirection operator
- โปรแกรม 11.1 นั้นเป็นเพียงตัวอย่างแสดงให้เห็นวิธีใช้ pointer แบบง่าย และไม่ใช่การใช้ pointer เพื่อเพิ่มประสิทธิภาพของโปรแกรม

Program 11.2 More Pointer Basics

```
// Further examples of pointers
#include <stdio.h>
int main (void)
{
    char c = 'Q';
    char *char_pointer = &c;

    printf ("%c %c\n", c, *char_pointer);

    c = '/';
    printf ("%c %c\n", c, *char_pointer);

    *char_pointer = '(';
    printf ("%c %c\n", c, *char_pointer);

    return 0;
}
```

```
char *char_pointer;

char_pointer = &c;
```

```
Q  Q
/  /
(  (
```

- ตัวแปร character ชื่อ c ถูกใส่ค่าเริ่มต้นให้เป็นตัวอักษร ‘Q’
- ตัวแปร pointer ชื่อ char_pointer ถูกประกาศให้เป็นชนิด char หมายความว่าข้อมูลที่ถูกชี้โดยตัวแปรนี้ จะเป็นชนิด char
- ในบรรทัดเดียวกันนี้มีการใส่ค่าเริ่มต้นให้กับ char_pointer เป็น address ของ c โดยใช้ address operator ‘&’ ดังนั้น char_pointer จะชี้ไปที่ c
- คำสั่ง printf() ครั้งแรก เรียกเพื่อแสดงตัวอักษรที่เก็บอยู่ในตัวแปร c และ ค่าที่อยู่ในตัวแปรที่ถูกชี้โดย char_pointer ซึ่งในที่นี้ก็คือค่าในตัวแปร c นั่นเอง
- บรรทัดต่อไป ตัวอักษร ‘/’ ถูกใส่ให้กับตัวแปร c และ char_pointer ยังชี้ไปที่ตัวแปรนี้อยู่ ทำให้คำสั่ง printf(..., *char_pointer) ในบรรทัดถัดมาแสดงตัว ‘/’ ออกทางหน้าจอ
- การทำคำสั่งบรรทัดถัดมา *char_pointer = ‘(’; หมายความว่า ให้นำตัวอักษร ‘(’ ไปเก็บในตัวแปรที่ถูกชี้โดย char_pointer นั่นก็คือ ‘(’ จะถูกเก็บลงในตัวแปร c นั่นเอง

9.2 การใช้ pointer ในการคำนวณ

- ในโปรแกรมถัดไป เราประกาศ pointer p1 และ p2 เพื่อชี้ตัวแปรชนิด integer

Program 11.3 Using Pointers in Expressions

```
// More on pointers
#include <stdio.h>
int main (void)
{
    int i1, i2;
    int *p1, *p2;

    i1 = 5;
    p1 = &i1;

    i2 = *p1 / 2 + 10;

    p2 = p1;

    printf("i1 = %i, i2 = %i, *p1 = %i, *p2 = %i\n", i1, i2,
*p1, *p2);

    return 0;
}
```

```
i1 = 5, i2 = 12, *p1 = 5, *p2 = 5
```

- โปรแกรมนี้ ประกาศตัวแปร i1 และ i2 เป็น integer และตัวแปร pointer p1 และ p2 จากนั้นให้ i1 เก็บค่า 5 และ p1 เก็บ address ของ i1
- i2 = *p1 / 2 + 10 นั่นคือ นำค่าในตัวแปรที่ชี้โดย p1 มาคำนวณ จึงมีความหมายเทียบเท่า i2 = i1 / 2 + 10 ซึ่งเท่ากับ _____
- ในคำสั่งถัดมา ค่า address ของ i1 ที่ถูกเก็บอยู่ใน p1 ได้ถูกคัดลอกลงใน p2 ดังนั้นทั้ง p1 และ p2 ชี้ไปที่ตัวแปรตัวเดียวกันนั่นคือ i1
- คำสั่ง printf แสดงผลค่า i1, i2, และ *p1, *p2

9.3 การใช้ pointer เพื่อชี้ structure

- เราได้เห็นการใช้ pointer เพื่อชี้ตัวแปรแบบปกติเช่น int และ char แล้ว และ pointer ยังสามารถใช้ชี้ตัวแปร structure ได้ด้วย
- จากบทที่ผ่านมาเราได้ประกาศ structure ชื่อ struct date

```
struct date
{
        int month;
        int day;
        int year;
};
```

- และประกาศตัวแปรเป็นชนิด struct date

```
struct date todaysDate;
```

- เราสามารถสร้าง pointer ขึ้นมาชี้ตัวแปร todaysDate ได้

```
struct date *datePtr;
datePtr = &todaysDate;
```

- จากนั้นเราสามารถใช้ pointer เพื่อเข้าถึงข้อมูลใน structure

```
(*datePtr).day = 21;
```

ซึ่งมีความหมายเดียวกับ

```
todaysDate.day = 21;
```

- หรือ

```
if( (*datePtr).month == 12 )
```

ซึ่งมีความหมายเทียบเท่า

```
if( todaysDate.month == 12)
```

- ในภาษา C มีการใช้ pointer กับ structure บ่อยมาก จึงมีการนิยาม operator พิเศษ '->' เรียก structure pointer operator ซึ่งทำให้เราสามารเขียน (*x).y ได้ในรูปแบบ x->y
- ดังนั้น if((*datePtr).month == 12) เขียนได้ใหม่เป็น if(datePtr->month == 12)
- เราเขียนโปรแกรม 9.1 ในเรื่องของ structure ใหม่ โดยใช้ pointer เข้ามาช่วยได้ดังนี้

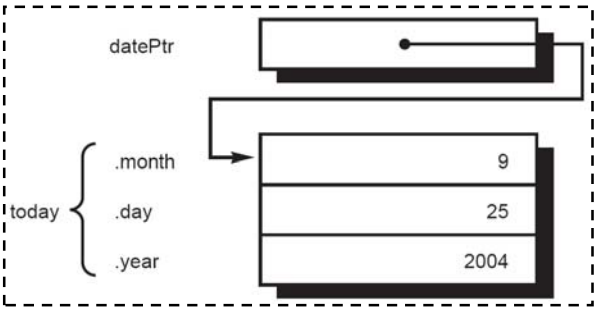
Program 11.4 Using Pointers to Structures

```
// Program to illustrate structure pointers
#include <stdio.h>
int main (void)
{
    struct date
    {
        int month;
        int day;
        int year;
    };
    struct date today, *datePtr;

    datePtr = &today;

    datePtr->month = 9;
    datePtr->day = 25;
    datePtr->year = 2004;

    printf ("Today's date is %i/%i/%.2i.\n",
        datePtr->month, datePtr->day, datePtr->year % 100);
    return 0;
}
```




```
Today's date is 9/25/04.
```

- สังเกตว่า pointer จะเก็บ address เริ่มต้นของตัวแปรย่อยตัวแรก

9.4 structure ที่มี pointer เป็นตัวแปรย่อย

- pointer สามารถเป็นตัวแปรย่อยอยู่ใน structure ได้ ดังเช่น

  struct intPtrs
  {
  int *p1;
  int *p2;
  };
  struct intPtrs x;

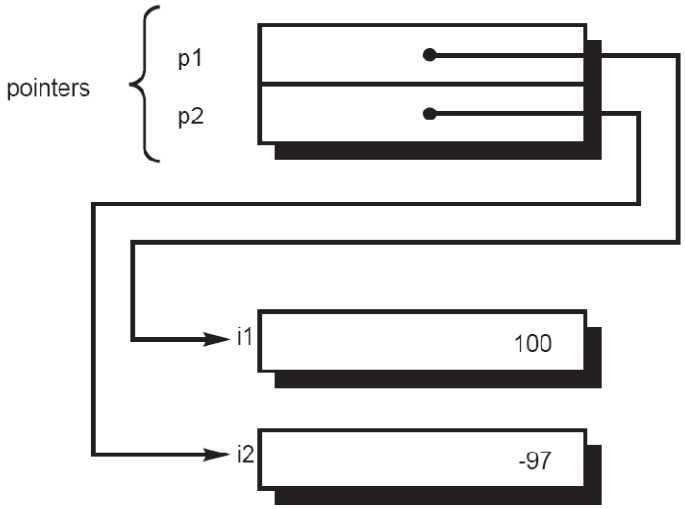
  จะเห็นว่า structure ชื่อ struct intPtrs ประกอบด้วยตัวแปรย่อยคือ pointer สองตัว สำหรับชี้ตัวแปร integer และเราสามารถประกาศตัวแปรชื่อ x ให้เป็นชนิด struct intPtrs ได้

Program 11.5 Using Structures Containing Pointers

```
// Function to use structures containing pointers
#include <stdio.h>
int main (void)
{
    struct intPtrs
    {
        int *p1;
        int *p2;
    };
    struct intPtrs x;
    int i1 = 100, i2;
    x.p1 = &i1;
    x.p2 = &i2;
    *x.p2 = -97;
    printf("i1 = %i, *x.p1 = %i\n", i1, *x.p1);
    printf("i2 = %i, *x.p2 = %i\n", i2, *x.p2);
    return 0;
}
```

```
i1 = 100, *x.p1 = 100
i2 = -97, *x.p2 = -97
```

- ในโปรแกรมนี้ address ของ i1 ถูกใส่ให้กับ pointer p1 ซึ่งเป็นสมาชิกย่อยของตัวแปร structure ชื่อ x ดังนั้น x.p1 จะชี้ไปที่ i1
- และเช่นเดียวกัน address ของ i2 ถูกใส่ให้กับ pointer p2 ดังนั้น x.p2 จะชี้ไปที่ i2
- จากนั้น *x.p2 = - 97; เป็นการใส่ค่าให้ตัวแปรที่ถูกชี้โดย x.p2 นั่นก็คือ i2 ในกรณีนี้ไม่ต้องใส่วงเล็บกำกับ เพราะเครื่องหมายจุด (structure member operator) มีความสำคัญสูงกว่า * (indirection operator)
- คำสั่ง printf บรรทัดแรก แสดงค่าที่เก็บใน i1 และค่าในตัวแปรที่ถูกชี้โดย x.p1
- คำสั่ง printf บรรทัดถัดมา แสดงค่าที่เก็บใน i2 และค่าในตัวแปรที่ถูกชี้โดย x.p2



9.5 การใช้ pointer กับฟังก์ชัน

- เราสามารถส่ง address ของตัวแปรเป็น argument ให้กับฟังก์ชันได้ โดยตัวแปรที่มารับค่า address ในฟังก์ชันก็จะต้องเป็นตัวแปร pointer
- การเปลี่ยนค่าของตัวแปรที่ถูกชี้โดย pointer ภายในฟังก์ชันก็จะมีผลกับตัวแปรที่อยู่นอกฟังก์ชันด้วย
- ดังนั้นการส่งผ่าน address เข้าไปในฟังก์ชันจะมีประโยชน์ในกรณีที่ต้องการใช้ฟังก์ชันเปลี่ยนแปลงค่าในตัวแปรหลายๆตัว ซึ่งการใช้งานปกติของฟังก์ชันจะส่งค่าคืนออกมาได้ค่าเดียวเท่านั้น (คำสั่ง return)

Program 11.8 Using Pointers and Functions

```
// Program to illustrate using pointers and functions
#include <stdio.h>

void test(int *int_pointer)
{
    *int_pointer = 100;
}

int main(void)
{
    int i = 50, *p = &i;

    printf("Before the call to test i = %i\n", i);
    test(p);
    printf("After the call to test i = %i\n", i);

    return 0;
}
```

```
Before the call to test i = 50
After the call to test i = 100
```

- ฟังก์ชันชื่อ test ถูกประกาศให้รับ argument หนึ่งตัวเป็น pointer ที่ชี้ไปที่ตัวแปร integer โดยการทำงานภายในฟังก์ชันคือการใส่ค่า 100 ลงในตัวแปร integer ที่ถูกชี้โดย pointer นั้น
- ฟังก์ชัน main ประกาศตัวแปรสองตัวคือ i ซึ่งเป็นตัวแปร integer เก็บค่าเริ่มต้น 50 และตัวแปร p ซึ่งเป็น pointer ที่เก็บค่าเริ่มต้นเป็น address ของตัวแปร i ดังนั้นตัวแปร p จะชี้ไปที่ตัวแปร i นั่นเอง
- จากนั้นโปรแกรมแสดงค่า i ก่อนการเรียกใช้งานฟังก์ชัน test
- ฟังก์ชัน test ถูกเรียกใช้งานโดยส่ง pointer p ซึ่งเก็บ address ของ i เข้าไปในฟังก์ชัน ซึ่ง address นี้ถูกคัดลอกลงในตัวแปร pointer ตัวใหม่ชื่อ int_pointer
- จากนั้นฟังก์ชัน test ใส่ค่า 100 ให้กับตัวแปรที่ถูกชี้โดย int_pointer ก็คือตัวแปร i ถูกเปลี่ยนค่าให้เป็น 100 จากภายในฟังก์ชัน
- ดังนั้นในขณะที่เรียกใช้ฟังก์ชัน ตัวแปร i ถูกชี้โดย pointer สองตัว คือ pointer ชื่อ p ที่อยู่นอกฟังก์ชัน และ pointer ชื่อ int_pointer ซึ่งอยู่ภายในฟังก์ชัน
- ตัวอย่างถัดไปเป็นการใช้ pointer เพื่อช่วยสลับค่าที่เก็บอยู่ในตัวแปรสองตัว

Program 11.9 Using Pointers to Exchange Values

```
// More on pointers and functions
#include <stdio.h>
void exchange(int * const pint1, int * const pint2)
{
    int temp;
    temp = *pint1;
    *pint1 = *pint2;
    *pint2 = temp;
}

int main(void)
{
    int i1 = -5, i2 = 66, *p1 = &i1, *p2 = &i2;

    printf("i1 = %i, i2 = %i\n", i1, i2);

    exchange(p1, p2);
    printf ("i1 = %i, i2 = %i\n", i1, i2);

    exchange(&i1, &i2);
    printf ("i1 = %i, i2 = %i\n", i1, i2);

    return 0;
}
```

```
i1 = -5, i2 = 66
i1 = 66, i2 = -5
i1 = -5, i2 = 66
```

- ฟังก์ชัน exchange ใช้สลับค่าเลขจำนวนเต็มที่เก็บในตัวแปร integer สองตัว โดยฟังก์ชันรับ argument เป็น pointer ที่ชี้ไปยังตัวแปร integer สองตัวนั้น
- การทำงานของฟังก์ชัน exchange ขั้นแรกจะนำค่าในตัวแปรที่ถูกชี้โดย pint1 ไปเก็บในตัวแปร temp ซึ่งเป็น local variable จากนั้นจะนำค่าในตัวแปรที่ถูกชี้

โดย pint2 ไปใส่ในตัวแปรที่ถูกชี้โดย pint1 และสุดท้ายจะนำค่าใน temp ไปใส่ในตัวแปรที่ถูกชี้โดย pint2

- ฟังก์ชัน main ประกาศตัวแปร integer สองตัวชื่อ i1 และ i2 เก็บค่า -5 และ 66 ตามลำดับ จากนั้น pointer ชื่อ p1 และ p2 ถูกเซ็ทให้ชี้ไปที่ตัวแปร i1 และ i2 ตามลำดับ
- printf บรรทัดแรกแสดงค่าของ i1 และ i2
- จากนั้นฟังก์ชัน exchange จึงถูกเรียกใช้โดยส่งค่า p1 และ p2 ให้เป็น argument โดยฟังก์ชันจะสลับค่าในตัวแปรที่ถูกชี้โดย p1 กับค่าในตัวแปรที่ถูกชี้โดย p2 นั่นก็คือค่าใน i1 และ i2 จะถูกสลับกัน
- printf บรรทัดถัดมาแสดงค่า i1 และ i2 ที่ถูกสลับแล้ว
- ฟังก์ชัน exchange ถูกเรียกใช้อีกครั้งหนึ่ง โดยส่ง address ของ i1 และ i2 เข้าไปแทน ซึ่งจะมีผลเหมือนกับการเรียกใช้ exchange ในครั้งแรก คือสลับค่าที่เก็บอยู่ใน i1 และ i2
- สังเกตว่าถ้าไม่ใช้เรื่องของ pointer เข้ามาช่วย เราจะเขียนฟังก์ชันเพื่อสลับค่าที่เก็บในตัวแปรสองตัวไม่ได้เลย เนื่องจากฟังก์ชันอนุญาตให้มีการส่งค่ากลับออกมาจากฟังก์ชันเพียงค่าเดียวเท่านั้น (ด้วยคำสั่ง return)

- นอกจากจะรับ address เข้าเป็น argument ของฟังก์ชันแล้ว ฟังก์ชันสามารถส่ง address ของตัวแปรคืนออกมาได้เช่นเดียวกัน

9.5 การใช้ pointer กับ array

- Pointer ถูกใช้ชี้ตัวแปร array เพื่อความสะดวกและประสิทธิภาพ โดยมีผลทำให้ใช้หน่วยความจำน้อยลงและทำงานได้ไวขึ้น
- สมมติว่าเรามี array ของ integer ที่มีสมาชิก 100 ตัวชื่อ values เราสามารถสร้าง pointer ชื่อ valuesPtr ขึ้นมาเพื่อใช้เข้าถึงสมาชิกใน array นี้ได้ โดยเราประกาศดังนี้

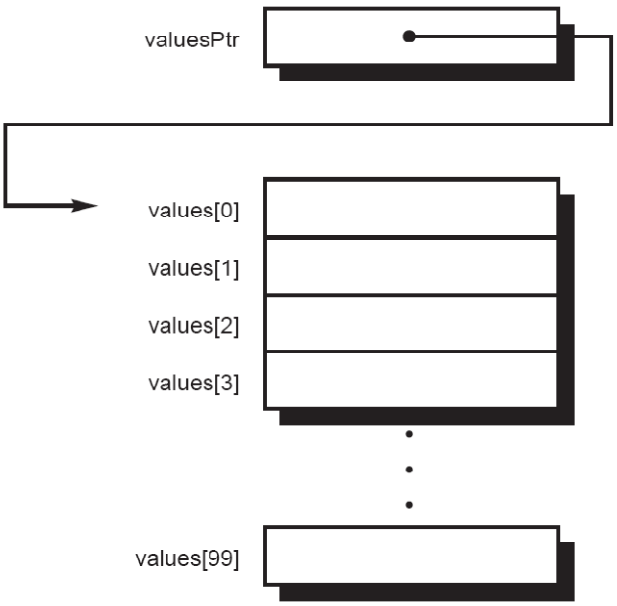
  ```
  int values[100];
  int *valuePtr;
  ```
- สังเกตว่าเมื่อเราประกาศ pointer สำหรับ array จะประกาศเพียงแค่ pointer เดียวสำหรับชี้ไปยังสมาชิกตัวแรกของ array เท่านั้น
- เพื่อที่จะให้ valuePtr ชี้ไปยังสมาชิกตัวแรก คือ values[0] จะใช้คำสั่ง

  ```
  valuePtr = values;
  ```

  โดยในการนี้จะไม่ใช้ address operator วางหน้าตัวแปร array เนื่องจากในภาษา C การอ้างชื่อ array โดยไม่มี index ต่อท้ายหมายถึงการอ้างถึง address ของสมาชิกตัวแรกใน array
- การทำดังนี้ มีผลเทียบเท่ากับคำสั่ง

  ```
  valuePtr = &values[0];
  ```

  ซึ่งหมายถึงการเอา address ของสมาชิกตัวแรกใน array ไปใส่ใน pointer เช่นเดียวกัน



- การใช้ pointer กับ array ทำให้เราสามารถเข้าถึงสมาชิกแต่ละตัวของ array ได้รวดเร็วขึ้น
- จากรูปจะเห็นว่า valuesPtr ชี้ที่สมาชิกตัวแรกของ array คือ values[0] ดังนั้น *valuePtr จะอ้างอิงไปที่ข้อมูลที่เก็บอยู่ในสมาชิกตัวแรกนั้น
- การอ้างอิงสมาชิกตัวถัดๆไปโดยใช้ pointer ที่กำลังชี้ไปที่สมาชิกตัวแรก ทำได้โดยบวกจำนวนเต็มเข้าไป เช่น *(valuesPtr + 3) หมายถึงข้อมูลใน values[3] และ *(valuesPtr + i) หมายถึงข้อมูลใน values[i]
- ดังนั้นเพื่อใส่ค่า 27 ให้กับ value[10] ทำได้โดยคำสั่ง

  values[10] = 27;

  หรือ

  *(valuesPtr + 10) = 27;
- การทำให้ pointer ชี้ไปที่สมาชิกตัวอื่นๆ อาจจะใช้คำสั่ง

  valuePtr = &values[i];

- และหาก pointer กำลังชี้ไปที่ values[0] เราสามารถเปลี่ยนให้ pointer ชี้ไปที่สมาชิกตัวถัดไปได้โดยใช้คำสั่ง

  valuesPtr += 1;
- เราสามารถเปรียบเทียบค่าที่เก็บใน pointer กับ address ของสมาชิกของ array ได้ เพื่อทดสอบว่า pointer ชี้เกินจำนวนสมาชิกใน array แล้วหรือยัง เช่น

  valuesPtr > &values[99]
  ซึ่งนิพจน์นี้จะเป็น TRUE เมื่อ pointer ชี้เกินสมาชิก 100 ตัวแล้ว
- โปรแกรมถัดไปเป็นการใช้ pointer กับ array

Program 11.11 Working with Pointers to Arrays

```
// Function to sum the elements of an integer array
#include <stdio.h>
int arraySum(int a[], const int n)
{
    int sum = 0, *ptr;
    for( ptr = a; ptr < a + n; ++ptr )
        sum += *ptr;
    return sum;
}

int main (void)
{
    int values[10] = { 3, 7, -9, 3, 6, -1, 7, 9, 1, -5 };
    printf ("The sum is %i\n", arraySum(values, 10));
    return 0;
}
```

```
The sum is 21
```

- ในฟังก์ชัน arraySum ลูป for ใช้เพื่อวนเข้าถึงสมาชิกแต่ละตัวใน array เพื่อนำค่าทั้งหมดที่เก็บใน array มาบวกลงในตัวแปร sum

- ในการนี้โปรแกรมใช้ pointer ชื่อ ptr เริ่มชี้จากสมาชิกตัวแรกของ array และในแต่ละครั้งที่วนลูปจะใช้ indirection operator เพื่อเข้าถึงสมาชิกของ array ที่ถูกชี้โดย ptr จากนั้น ptr จะถูกเพิ่มค่าขึ้นหนึ่งค่าเพื่อชี้สมาชิกตัวถัดไปของ array ลูป for จะวนจนครบจำนวนสมาชิกของ array โดยตรวจสอบเงื่อนไข ptr < a+n นั่นคือ address ที่ ptr เก็บไว้ จะต้องน้อยกว่าหรือเท่ากับ address ของสมาชิกตัวสุดท้ายของ array