ជំពូកទី ៧

អំពី Pointers

១. <u>សេចក្តីផ្តើម</u>

ជំហានដំបូងដើម្បីយល់បានពី Pointer នោះគឺត្រូវមើលឲ្យឃើញនូវការតាងក្នុងកម្រិតម៉ាស៊ីន។ កុំព្យូទ័រទំនើបភាគច្រើន memory ចំបងត្រូវបានបែងចែកជា bytes ដែលក្នុងមួយ byte អាចផ្ទុកព័ត៌មាន 8 bits :

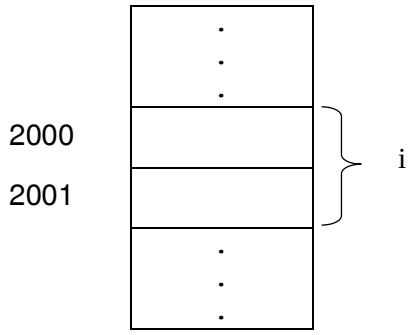
0	1	0	1	0	0	1	1

ចំពោះ byte និមួយៗមានអស័យដ្ឋានមួយដើម្បីសម្គាល់លក្ខណៈផ្សេងគ្នាពី bytes ផ្សេងទៀតក្នុង memory។ បើសិនជាវាមានចំនួន n bytes ក្នុង memory គេអាចគិតពីអស័យដ្ឋានដូចទៅនឹងលេខដែលមានតម្លៃពីសូន្យទៅ n-1 (សូមមើលរូបតាងអស័យដ្ឋាននៅខាងក្រោមនេះ)។

កម្មវិធីប្រតិបត្តិរួមមានទាំង code និង data (អញ្ញាតក្នុងកម្មវិធី)។ អញ្ញាតនិមួយៗក្នុងកម្មវិធី ប្រើមួយ byte ឬ ច្រើន bytes ក្នុង memory នោះ អស័យដ្ឋានរបស់ byte ទី១ ត្រូវបាននិយាយថា ជាអស័យដ្ឋានរបស់អញ្ញាត។

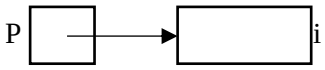
អស័យដ្ឋាន	តម្លៃ
0	01010011
1	01110101
2	01110011
3	01100001
4	01101110
	. . .
n-1	01000011

នៅក្នុងរូបខាងក្រោម អញ្ញាត i ប្រើចំនួន bytes នៅត្រង់អស័យដ្ឋាន 2000 និង 2001 ដូចនេះ អាស័យដ្ឋានរបស់ i គឺ 2000 :



នៅកន្លែងនេះដែល pointer ចូលប្រើត្រង់នេះ។ ទោះជាអស័យដ្ឋានតាងដោយលេខក៏ដោយ ដែនកំណត់នៃតម្លៃអាចខុសគ្នាពីចំនួនគត់នោះ ហេតុនេះគេមិនអាចផ្ទុកតម្លៃនៅក្នុងអញ្ញាតចំនួនគត់ធម្មតា។ ទោះជាយ៉ាងណាក៏ដោយ គេអាចផ្ទុកវានៅក្នុងអញ្ញាត pointer ពិសេស។ នៅពេលគេផ្ទុកអស័យដ្ឋានរបស់អញ្ញាត i មួយនៅក្នុង អញ្ញាត pointer ឈ្មោះ p គេអាចនិយាយថា p "points to" i។ គេអាចនិយាយម្យ៉ាងទៀតថា pointer មួយគឺគ្មានអ្វីលើសពីអស័យដ្ឋានមួយ ហើយអញ្ញាត pointer មួយ គឺគ្រាន់តែជាអញ្ញាតមួយដែលអាចផ្ទុកអស័យដ្ឋានមួយ។

ជំនួសឲ្យការបង្ហាញអស័យដ្ឋានដូចទៅនឹងលេខនៅក្នុងឧទាហរណ៍របស់យើង។ ដើម្បីបង្ហាញអញ្ញាត pointer មួយ ឈ្មោះ p ផ្ទុកអស័យដ្ឋានរបស់អញ្ញាត i នោះ គេអាចបង្ហាញតម្លៃរបស់ p ដូចទៅនឹងសញ្ញាព្រួញ មានទិសដៅឆ្ពោះទៅរក i ៖



២. <u>ការប្រកាសអញ្ញត</u> Pointer

អញ្ញាត pointer មួយត្រូវបានប្រកាសដូចគ្នាច្រើនទៅនឹងការប្រកាសអញ្ញាតធម្មតាដែរ។ ភាពខុសគ្នាត្រង់ថា ឈ្មោះអញ្ញាត pointer មួយត្រូវតែដាក់សញ្ញា * នៅពីមុខអញ្ញាត៖

```
int *p;
```

ការប្រកាសនេះ ប្រាប់ឲ្យដឹងថា p គឺជាអញ្ញាត pointer មួយដែលអាចចង្អុលទៅកាន់ objects នៃប្រភេទ int ។ ក្នុងនេះ object ជាពាក្យប្រើជំនួសឲ្យអញ្ញាត។

អញ្ញាត pointer អាចលេចឡើងនៅក្នុងការប្រកាសជាមួយអញ្ញាតផ្សេងទៀត៖

```
int i, j, a[10], b[20], *p, *q;
```

នៅក្នុងឧទាហរណ៍ខាងលើនេះ i និង j គឺជាអញ្ញាតចំនួនគត់ធម្មតា ហើយ a និង b គឺជា arrays នៃចំនួនគត់ រីឯ p និង q គឺជាអញ្ញាត pointers ដែលចង្អុលទៅកាន់អញ្ញាតចំនួនគត់។

ភាសា C តម្រូវឲ្យអញ្ញាត pointer ចង្អុលទៅកាន់តែចំពោះអញ្ញាតនៃប្រភេទត្រូវគ្នាប៉ុណ្ណោះ៖

```
int *p;   /* points only to integers */
double *q;  /* points only to doubles */
char *r;  /* points only to characters */
```

៣. អំពីសញ្ញាណពុន្ត & (address operator) និង សញ្ញាណពុន្ត * (indirection operator)

ភាសា C ផ្តល់នូវសញ្ញាណពុន្តមួយគូ ដែលរៀបចំឡើងសម្រាប់ប្រើជាមួយ pointers។ ដើម្បីរកអស័យដ្ឋាននៃអញ្ញាតមួយ គេប្រើ សញ្ញាណពុន្ត & ។ បើ x ជាអញ្ញាតមួយ ពេលនោះ &x គឺជាអស័យដ្ឋាននៃ x ក្នុង memory។ ដើម្បីចូលប្រើអញ្ញាតដែល pointer មួយចង្អុលទៅកាន់នោះ គេប្រើសញ្ញាណពុន្ត * ។ បើ p គឺជា pointer ពេលនោះ *p តាងឲ្យអញ្ញាត ដែល p កំពុងចង្អុលទៅរក។

៣.១ សញ្ញាណពុន្តអស័យដ្ឋាន & (Address operator)

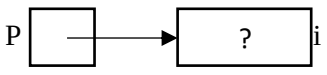
ការប្រកាសអញ្ញាត pointer មួយកំណត់ទំហំមួយសម្រាប់ pointer ប៉ុន្តែមិនធ្វើការចង្អុលទៅកាន់អញ្ញាតណាមួយនៅឡើយ :

```
int *p;   /* points nowhere in particular */
```

វាមានសារៈសំខាន់ដើម្បីកំណត់តម្លៃដំបូង p មុននឹងប្រើវា។ វិធីមួយដើម្បីកំណត់តម្លៃដំបូងឲ្យអញ្ញាត pointer គឺកំណត់អស័យដ្ឋាននៃអញ្ញាតទៅឲ្យវា ឬ ជាទូទៅ គេកំណត់តម្លៃឲ្យវាដោយប្រើសញ្ញាណពុន្ត & :

```
int i, *p;
...
p = &i;
```

ដោយកំណត់អស័យដ្ឋានរបស់ i ទៅឲ្យអញ្ញាត p ឃ្លានេះធ្វើឲ្យ p ចង្អុលទៅកាន់ i :



គេក៏អាចប្រកាសអញ្ញាត pointer មួយដោយកំណត់តម្លៃដំបូងឲ្យវាក្នុងពេលតែមួយ៖

```
int i;
int *p = &i;
```

៣.២ <u>សញ្ញាណនព្វន្ត</u> * (Indirection operator)

កាលណា អញ្ញាត pointer មួយចង្អុលទៅកាន់អញ្ញាតមួយ គេអាចប្រើសញ្ញាណនព្វន្ត * ដើម្បីចូលប្រើនូវអ្វីដែលបានផ្ទុកក្នុងអញ្ញាតនោះ។ ឧទាហរណ៍ បើ p ចង្អុលទៅកាន់ i នោះ គេអាចបោះតម្លៃរបស់ i ដូចខាងក្រោម៖

```
printf("%d\n", *p);
```

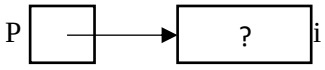
printf នឹងបង្ហាញតម្លៃរបស់ i , ដោយមិនបង្ហាញអស័យដ្ឋានរបស់ i ទេ។

ការប្រើ & ទៅលើអញ្ញាតមួយបង្កើតនូវ pointer មួយឲ្យអញ្ញាតនោះ ការប្រើ * ទៅលើ pointer នោះនាំឲ្យយើងបានអញ្ញាតដើមវិញ៖

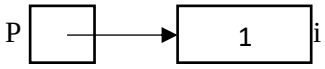
```
j = *&i;  /* same as j = i; */
```

ដរាបណា p ចង្អុលទៅកាន់ i នោះ *p គឺជាឈ្មោះមួយទៀតសម្រាប់ i។ វាមិនគ្រាន់តែ *p ដូច i ប៉ុណ្ណោះទេ ក៏ប៉ុន្តែការប្តូរតម្លៃ របស់ *p នោះធ្វើឲ្យប្តូរតម្លៃរបស់ i ផងដែរ។ ឧទាហរណ៍ខាងក្រោមបង្ហាញពីសមមូលនៃ *p និង i ។ ដ្យាក្រាមបង្ហាញតម្លៃរបស់ p និង i នៅត្រង់ចំណុចផ្សេងគ្នាក្នុងការគណនា។

```
p = &i;
```

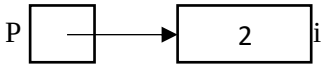



```
i = 1;
```




```
printf("%d\n", i);       /* prints 1 */
printf("%d\n", *p);      /* prints 1 */

*p = 2;
```




```
printf("%d\n", i);       /* prints 2 */
printf("%d\n", *p);      /* prints 2 */
```

គេមិនត្រូវប្រើសញ្ញាណនព្វន្ត * ទៅលើអញ្ញាត pointer ដែលមិនទាន់កំណត់តម្លៃឡើយ។ បើសិនអញ្ញាត pointer p មិនទាន់កំណត់តម្លៃដំបូង ហើយព្យាយាមប្រើតម្លៃរបស់ p ក្រោមរូបភាពបែបណាក៏ដោយ នោះវាបណ្តាលកើតឡើងនូវលក្ខណៈមិនកំណត់។ ឧទាហរណ៍ខាងក្រោមនេះ ការហៅ printf បោះតម្លៃសំណល់ក្នុង memory បណ្តាលឲ្យកម្មវិធីគាំង ឬមានផលវិបាកផ្សេងទៀតកើតឡើង:

```
int *p;
printf("%d", *p);    /*** wrong ***/
```

ការកំណត់តម្លៃមួយទៅឲ្យ *p គឺគ្រោះថ្នាក់ណាស់។ បើសិន p កើតឡើងបានផ្ទុកនូវអស័យដ្ឋាន memory ត្រឹមត្រូវហើយនោះ ការកំណត់តម្លៃខាងក្រោមនឹងព្យាយាមកែប្រែទិន្នន័យដែលបានផ្ទុកនៅត្រង់អស័យដ្ឋាននោះ៖

```
int *p;
*p = 1;         /*** wrong ***/
```

បើសិនទីតាំងបានកែប្រែដោយសារការកំណត់តម្លៃនេះដែលជារបស់កម្មវិធី វាអាចមាន error កើតឡើង តែបើវាជារបស់ប្រព័ន្ធប្រតិបត្តិការនោះ កម្មវិធីនឹងគាំងភាគច្រើន។ compiler អាចបញ្ចេញការព្រមានថា p គឺមិនទាន់កំណត់តម្លៃដំបូង ដូចនេះយើងគួរយកចិត្តទុកដាក់ចំពោះសារព្រមានណាមួយកើតឡើងដែលយើងបានទទួល។

៤. **អំពី** Pointer assignment

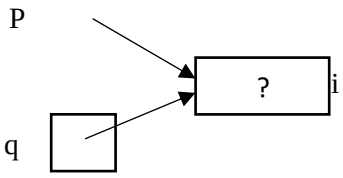
ភាសា C អាចឲ្យការប្រើសញ្ញាណនព្វន្ត = ចម្លង pointers ដោយវាផ្តល់ឲ្យបានចំពោះប្រភេទទិន្នន័យដូចគ្នា។ ឧបមាថា i, j, p និង q ត្រូវបានប្រកាសដូចខាងក្រោម៖

```
int i, j, *p, *q;
```

ឃ្លា p = &i; គឺជាឧទាហរណ៍នៃ pointer assignment ហើយអស័យដ្ឋានរបស់ i ត្រូវបានចម្លងទៅឲ្យ p ។ នេះជាឧទាហរណ៍មួយទៀតនៃ pointer assignment :

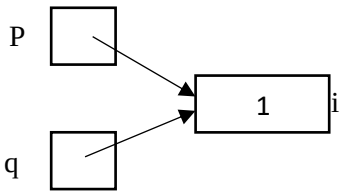
```
q = p;
```

ឃ្លានេះចម្លងតម្លៃរបស់ p (អស័យដ្ឋានរបស់ i) ទៅឲ្យ q ធ្វើឲ្យ q ចង្អុលទៅកាន់ទីតាំងដូចគ្នាទៅនឹង p ដែរ ៖

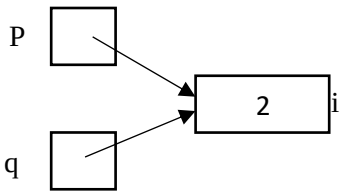


ពេលនេះទាំង p និង q ចង្អុលទៅកាន់ i ដូច្នេះយើងអាចប្តូរ i ដោយកំណត់តម្លៃថ្មីទៅឲ្យទាំង *p ឬ *q :

*p = 1;



*q = 2;



ចំនួនអញ្ញាត pointers ណាក៏ដោយអាចចង្អុលទៅរក object ដូចគ្នា។ គួរយកចិត្តទុកដាក់ កុំច្រឡំ

q = p;

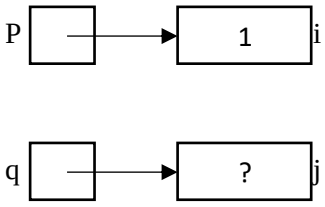
ជាមួយនឹង

*q = *p;

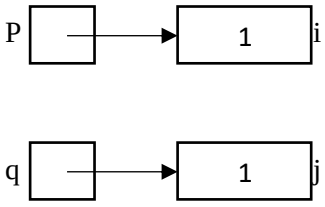
ឃ្លាទី១ គឺជា pointer assignment រីងឃ្លាទី ២ មិនមែនទេ ដូចបានបង្ហាញក្នុងឧទាហរណ៍ខាងក្រោមនេះ ៖

```
p = &i;
q = &j;
```

i = 1;



*q = *p;



ការកំណត់តម្លៃ *q = *p នឹងចម្លងតម្លៃដែល p ចង្អុលទៅកាន់ (តម្លៃរបស់ i) object ដែល q ចង្អុលទៅកាន់ (អញ្ញាត j)។

៥. ការប្រើ Pointer ជាមួយនឹងអនុគមន៍

៥.១ ការប្រើ Pointer ជា arguments

នៅក្នុងផ្នែកនេះ យើងនឹងឃើញពីសារៈសំខាន់នៃការប្រើអញ្ញាត pointer ជា argument នៃ អនុគមន៍។ ការប្រើអញ្ញាត pointer ជា argument របស់អនុគមន៍មួយអាចឲ្យគេកែប្រែតម្លៃរបស់អញ្ញាត។

ឧទាហរណ៍ គេចង់ធ្វើការបណ្តូរនៃតម្លៃពីរអញ្ញាតដោយប្រើ pointer ជា argument នៃអនុគមន៍ដូចក្នុងអនុគមន៍ខាងក្រោម៖

```
#include <conio.h>
#include <stdio.h>
void swap(int *a, int *b){
  int c;
  c = *a;
  *a = *b;
  *b = c;
}
void main() {
   int x=5, y=7; // ការប្រកាសនិងកំណត់តម្លៃឲ្យអញ្ញាត x និង y ជាប្រភេទ int
   printf("x = %d\t y = %d\n", x, y);
```

```
                    // បង្ហាញតម្លៃរបស់អញ្ញាត x និង y មកលើអេក្រង់
    swap(&x, &y); // ការហៅអនុគមន៍ swap() មកប្រើ
    printf("x = %d\t y = %d\n", x, y);
                // បង្ហាញតម្លៃរបស់អញ្ញាត x និង y មកលើអេក្រង់
    getch();
}
```

ឧទាហរណ៍ខាងក្រោមនេះបង្ហាញពីការប្រើអញ្ញាត pointer ជា argument នៃអនុគមន៍រកតម្លៃអតិបរមានិងអប្បបរមាក្នុង array មួយវិមាត្រផ្ទុកចំនួនគត់។ តម្លៃអតិបរមានិងអប្បបរមាត្រូវបញ្ជូនមកក្រៅតាម argument របស់អនុគមន៍។

```
#include <conio.h>
#include <stdio.h>
void maxmin(int a[], int n, int *max, int *min){
  int i;   // ការប្រកាសអញ្ញាត i ជាប្រភេទ int
  *max = *min = a[0];
  for (i = 1; i<n; i++){
     if (a[i] > *max)   // បើ a[i] > *max នោះ
       *max = a[i];      // កំណត់តម្លៃ a[i] ទៅឲ្យអញ្ញាត *max
     else if (a[i] < *min)  // បើ a[i] < *min នោះ
       *min = a[i];      // កំណត់តម្លៃ a[i] ទៅឲ្យអញ្ញាត *max
  }
}
void main() {
    int c[10],i,big,small;  // ការប្រកាសអញ្ញាត c[ ], i, big, small ជាប្រភេទ int
    printf("Enter 10 numbers : ");
    for (i=0; i < 10; i++)
      scanf("%d", &c[i]); // ទទួលតម្លៃពី keyboard និងផ្ទុកឲ្យអញ្ញាត c[i]
    maxmin(c, 10, &big, &small); // ហៅអនុគមន៍ maxmin() មកប្រើ
    printf("Largest : %d\n", big);
                    // បង្ហាញតម្លៃរបស់អញ្ញាត big មកលើអេក្រង់
```

```
        printf("Smallest : %d\n", small);
                        // បង្ហាញតម្លៃរបស់អញ្ញាត small មកលើអេក្រង់
        getch();
    }
```

៥.២ ការប្រើ Pointer ជា return តម្លៃ

យើងមិនត្រឹមតែអាចសរសេរ pointers បញ្ជូនទៅកាន់អនុគមន៍ប៉ុណ្ណោះទេ ក៏ប៉ុន្តែថែមទាំងយើងក៏អាចសរសេរអនុគមន៍ឲ្យតម្លៃជា pointers បានដែរ។

អនុគមន៍ខាងក្រោមនេះ នឹងឲ្យតម្លៃជា pointer មួយដែលមានតម្លៃជាចំនួនគត់ធំជាង កាលណា pointers ដែលគេឲ្យបញ្ជូនចូលជាពីរចំនួនគត់ ៖

```
int* max(int *a, int *b){
   if (*a > *b)   // បើ *a > *b នោះ
      return a;  // ឲ្យតម្លៃរបស់អញ្ញាត a
   else
      return b;  // ឲ្យតម្លៃរបស់អញ្ញាត b
}
void main() {
  int x=20, y=10;  // ការប្រកាសនិងកំណត់តម្លៃឲ្យអញ្ញាត x=20, y=10 ជាប្រភេទ int
  int *p;  // ការប្រកាសអញ្ញាត pointer p ជាប្រភេទ int
  p = max(&x, &y);  // ការកំណត់តម្លៃឲ្យអញ្ញាត p ដែលទទួលបានពីអនុគមន៍ max()
  printf("max = %d\n", *p);   // បង្ហាញតម្លៃរបស់អញ្ញាត *p មកលើអេក្រង់
  getch();
}
```

នៅពេលដែលយើងប្រើអនុគមន៍ max, យើងនឹងបញ្ជូន pointers ទៅឲ្យអញ្ញាតពីរជាចំនួនគត់ប្រភេទ int ហើយផ្ទុកលទ្ធផលក្នុងអញ្ញាត pointer មួយ។ ក្នុងពេលប្រើអនុគមន៍ max, *a គឺជាឈ្មោះមួយទៀត តាងឲ្យ x រីឯ *b គឺជាឈ្មោះមួយទៀតតាង y ។ បើសិនជា x មានតម្លៃធំជាង y នោះអនុគមន៍ max នឹងឲ្យតម្លៃជាអស័យដ្ឋានរបស់ x ផ្ទុយទៅវិញ វានឹងឲ្យតម្លៃជាអស័យដ្ឋានរបស់ y។ បន្ទាប់ពី ហៅអនុគមន៍មកប្រើ p ចង្អុលទៅកាន់ x ឬ y។

ទោះបីជាអនុគមន៍ max ឲ្យតម្លៃជា pointer មួយក្នុងចំណោម pointers ជាច្រើនដែលបានបញ្ជូនទៅកាន់វាតាម argument នោះមិនត្រឹមតែអាចធ្វើបានប៉ុណ្ណោះទេ ថែមទាំងវាអាចឲ្យអនុគមន៍ មួយឲ្យ

តម្លៃជា pointer មួយដែលចង្អុលទៅកាន់អញ្ញាតខាងក្រៅអនុគមន៍ឬអញ្ញាតក្នុងអនុគមន៍ដែល បានប្រកាសជាលក្ខណៈ static។

ផ្ទុយទៅវិញ គេមិនអាចឲ្យអនុគមន៍ឲ្យតម្លៃជា pointer មួយដែលចង្អុលទៅកាន់អញ្ញាតក្នុងអនុគមន៍ដែលមានលក្ខណៈ automatic ឡើយ៖

```
int *func(void){
   int i;
   ...
   return &i;
}
```

អញ្ញាត i មិនស្ថិតនៅទៀតទេ នៅពេលដែលអនុគមន៍ func ឲ្យតម្លៃ ហេតុនេះ pointer ចង្អុលទៅកាន់អញ្ញាតនេះមិនត្រឹមត្រូវឡើយ។ ចំពោះស្ថានភាពបែបនេះ compilers ខ្លះបង្ហាញសារអម្លឹកដូចជា "function returns address of local variable" ។

Pointers មិនគ្រាន់តែអាចចង្អុលទៅកាន់អញ្ញាតធម្មតាទេ ថែមទាំងអាចចង្អុលទៅកាន់ធាតុ array ផងដែរ។ បើសិន a ជា array មួយ ពេលនោះ &a[i] គឺជា pointer មួយនៅត្រង់ធាតុ i នៃ a ។ កាលណាអនុគមន៍មួយមាន argument ជា array មួយ ជួនកាលវាមានប្រយោជន៍សម្រាប់អនុគមន៍ ឲ្យតម្លៃជា pointer នៅត្រង់ធាតុណាមួយក្នុង array ។ ឧទាហរណ៍ អនុគមន៍ខាងក្រោមនេះឲ្យតម្លៃជា pointer មួយនៅត្រង់ធាតុកណ្តាលនៃ array a ដោយឧបមាថា a មាន n ធាតុ៖

```
int* find_middle(int a[], int n){
     return &a[n/2];
}
```

៦. <u>ការប្រើ</u> Pointer និង Array

នៅពេល array មួយត្រូវបានប្រកាស compiler បង្កើតទីតាំង memory នូវទំហំមួយគ្រប់គ្រាន់ដើម្បីផ្ទុកគ្រប់ធាតុនៃ array។ ផ្អែកលើអស័យដ្ឋានដែលផ្តល់នូវទីតាំងនៃធាតុទី១ ត្រូវបានកំណត់ទីតាំងដោយ compiler ផងដែរ។

ឧបមាថា យើងប្រកាស array មួយឈ្មោះ arr ៖

```
int arr[5] = {1, 2, 3, 4, 5};
```

ឧបមាថា អស័យដ្ឋានរបស់ arr គឺ 1000 ហើយចំនួនគត់និមួយៗត្រូវការ 2 bytes នោះធាតុទាំងប្រាំត្រូវបានផ្ទុកដូចទម្រង់ខាងក្រោម៖

នៅត្រង់នេះ អញ្ញាត arr នឹងផ្តល់នូវអស័យដ្ឋានសំខាន់ដែលជា pointer តម្លៃថេរ ចង្អុលទៅកាន់ធាតុ arr[0]។ ហេតុនេះ arr ត្រូវបានផ្ទុកអស័យដ្ឋានរបស់ arr[0] គឺ 1000។

arr គឺស្មើនឹង &arr[0] តាមលក្ខណៈ default។

យើងអាចប្រកាស pointer ប្រភេទ int ដើម្បីចង្អុលទៅកាន់ array arr។

```
int *p;      // ការប្រកាសអញ្ញាត *p ជាប្រភេទ int
p = arr;     // ការកំណត់តម្លៃជាអស័យដ្ឋានរបស់ arr ទៅឲ្យអញ្ញាត pointer
// or p = &arr[0];  // both the statements are equivalent.
```

ពេលនេះ យើងអាចចូលប្រើធាតុនិមួយៗរបស់ array arr ដោយប្រើ p++ ដើម្បីរំកិលពីធាតុមួយទៅធាតុមួយទៀត។

យើងអាចប្រើ pointer មួយចង្អុលទៅកាន់ array មួយ ពេលនោះយើងអាចប្រើ pointer នេះដើម្បីចូលប្រើ array។ ចូរសង្កេតឧទាហរណ៍ខាងក្រោមនេះ៖

```
int i;   // ការប្រកាសអញ្ញាត i ជាប្រភេទ int
int a[5] = {1, 2, 3, 4, 5}; // ការប្រកាស array a ដែលមាន 5 ធាតុ
int *p = a;  // same as int *p = &a[0]
for (i=0; i<5; i++){
  printf("%d", *p); // បង្ហាញតម្លៃរបស់អញ្ញាត *p មកលើអេក្រង់
  p++;   // បង្កើនតម្លៃ p មួយឯកតា
}
```

នៅក្នុងកម្មវិធីខាងលើ pointer *p នឹងបោះបង្ហាញគ្រប់តម្លៃទាំងអស់ដែលបានផ្ទុកក្នុង array ម្តងមួយៗ។ យើងក៏អាចប្រើអស័យដ្ឋានមូលដ្ឋាន (a ក្នុងកម្មវិធីខាងលើ) ដើម្បីធ្វើសកម្មភាពដូច pointer ហើយបោះបង្ហាញគ្រប់តម្លៃទាំងអស់។

យើងអាចជំនួសឲ្យឃ្លាក្នុងឧទាហរណ៍ខាងលើគឺ printf("%d", *p); ទៅនឹងឃ្លាដូចបានអធិប្បាយខាងក្រោម។ ចូរសង្កេតលទ្ធផលដែលទទួលបាន៖

```
printf("%d", a[i] );        // prints the array, by incrementing index
printf("%d", a+i );         // This will print address of all the array elements
printf("%d", *(a+i));       // It will print value of array element
printf("%d", *a);           // It will print value of a[0] only
```

ចំពោះការសរសេរ a++; នោះវានឹងនាំមក error ពេលធ្វើការ compile ព្រោះគេមិនអាចប្តូរអស័យដ្ឋានរបស់ array ឡើយ។

៧. **ការប្រើ** Pointer arithmetic

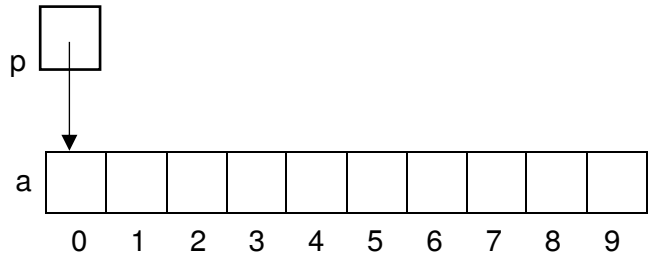
យើងបានឃើញហើយថា pointer អាចចង្អុលទៅកាន់ធាតុនៃ array។ ឧទាហរណ៍ ឧបមាថា a និង p ត្រូវបានប្រកាសដូចខាងក្រោម៖

```
int a[10], *p;
```

គេអាចធ្វើឲ្យ p ចង្អុលទៅកាន់ a[0] ដោយសរសេរ៖

```
p = &a[0];
```

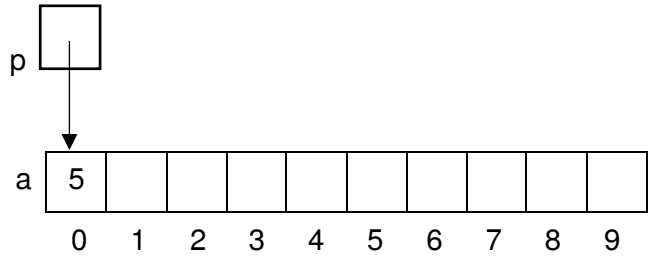
រូបភាពខាងក្រោមនេះបង្ហាញនូវការប្រើលក្ខណៈនេះ៖



ឥឡូវនេះ គេអាចចូលប្រើ a[0] តាមរយៈ p ឧទាហរណ៍គេអាចផ្ទុកតម្លៃ 5 ក្នុង a[0] ដោយសរសេរ ៖

```
*p = 5;
```

ដូចនេះ យើងអាចសរសេរ ៖



ការធ្វើឲ្យ pointer p មួយចង្អុលទៅកាន់ធាតុមួយនៃ array a មិនមែនឲ្យយើងយើញច្បាស់ តាមរូបភាពខាងក្រៅឡើយ។ ទោះជាយ៉ាងណាក៏ដោយ ការធ្វើប្រមាណវិធីជាមួយនឹង pointer (ឬអស័យដ្ឋានធ្វើប្រមាណវិធី) លើ p យើងអាចចូលប្រើធាតុដទៃទៀតរបស់ array a ។ ភាសា C ផ្តល់នូវការធ្វើប្រមាណវិធីជាមួយ pointer ចំនួន បីទម្រង់:

- បូកចំនួនគត់ (integer) ទៅឲ្យ pointer មួយ
- ដកចំនួនគត់ (integer) ពី pointer មួយ
- ដក pointer មួយ ពី pointer មួយទៀត

ចូរយើងពិនិត្យឲ្យបានច្បាស់នូវការធ្វើប្រមាណវិធីទាំងនេះ។ ឧទាហរណ៍ ឧបមាថា គេបានប្រកាសដូចខាងក្រោមនេះ៖

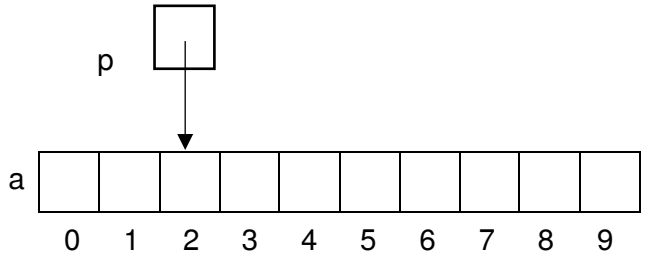
```
int a[10], *p, *q, i, j;
```

៧.១ បូកចំនួនគត់ (integer) ទៅឲ្យ pointer មួយ

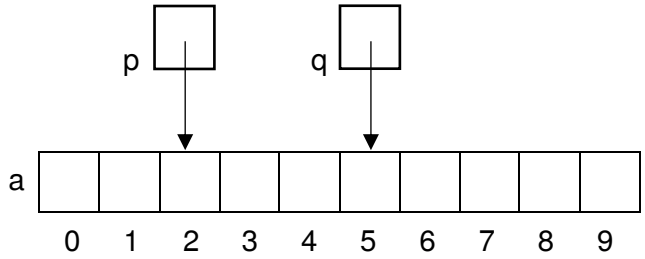
ការបូកចំនួនគត់ integer j មួយទៅឲ្យ pointer p បានធ្វើឲ្យ pointer មួយនៅត្រង់ធាតុ j បន្ទាប់ពីអញ្ជាត pointer ចង្អុលទៅកាន់ធាតុមួយ។ ដើម្បីឲ្យឃើញច្បាស់ជាងនេះទៀត បើ p ចង្អុលទៅកាន់ធាតុ array a[i] នោះ p + j ចង្អុលទៅកាន់ a[i + j]។

ឧទាហរណ៍ខាងក្រោមនេះ បង្ហាញ ការបូក pointer ដោយដ្យាក្រាមបង្ហាញតម្លៃរបស់ p និង q នៅត្រង់ចំណុចផ្សេងគ្នាក្នុងការគណនា។

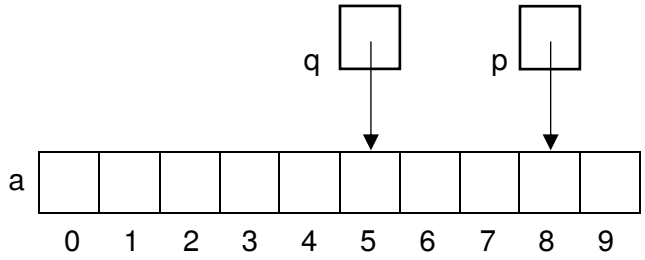
p = &a[2];



q = p + 3;



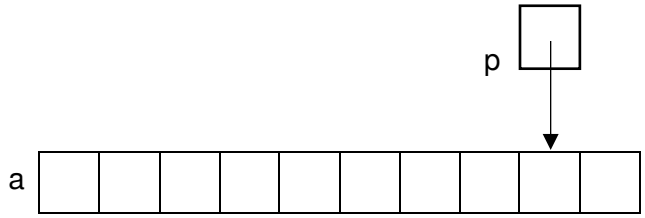
p += 6;



៧.២ ដកចំនួនគត់ (integer) ពី pointer មួយ

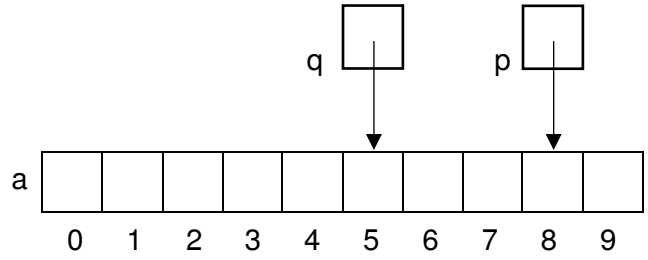
បើ p ចង្អុលទៅកាន់ធាតុ array ត្រង់ a[i] ពេលនោះ p – j ចង្អុលទៅកាន់ a[i – j] ។
ឧទាហរណ៍៖

p = &a[8];

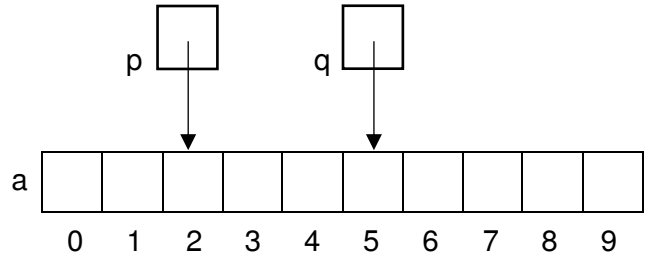


0 1 2 3 4 5 6 7 8 9

q = p - 3;



p –= 6;

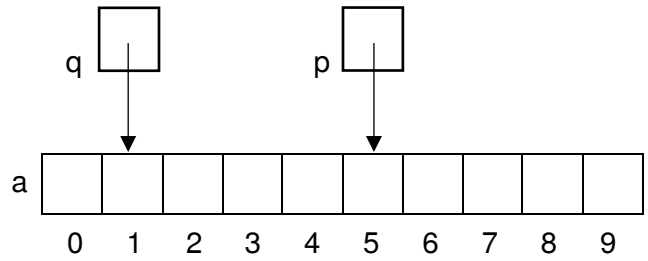


៧.៣ ដក pointer មួយ ពី pointer មួយទៀត

នៅពេលដែល pointer មួយត្រូវដកពី pointer មួយទៀត នោះលទ្ធផលជាចម្ងាយ (វាស់ក្នុងធាតុ array) រវាង pointer និង pointer។ ហេតុនេះ បើ p ចង្អុលទៅកាន់ a[i] និង q ចង្អុលទៅកាន់ a[j], ពេលនោះ p – q គឺស្មើនឹង i – j ។

ឧទាហរណ៍ ៖

```
p = &a[5];
q = &a[1];
```




```
i = p - q;      /* i is 4 */
```

```
i = q - p;      /* i is -4 */
```

ការធ្វើប្រមាណវិធីលើ pointer មួយដែលមិនបានចង្អុលទៅកាន់ធាតុ array បណ្តាលឲ្យការប្រព្រឹត្តមិនមានន័យ។ លើសពីនេះ លទ្ធផលនៃផលដក pointer មួយពី pointer មួយទៀតមិនមាន ន័យទេ ដរាបណា pointers ទាំងពីរចង្អុលទៅកាន់ធាតុនៃ array តែមួយ។

ក្រៅពីនេះ យើងអាចធ្វើការប្រៀបធៀប pointers ដោយប្រើសញ្ញាណនព្វន្តលេខធៀប (< , <=, >, >=) និង សញ្ញាណនព្វន្តលេខស្មើ (== និង !=)។ ការប្រើសញ្ញាណនព្វន្តលេខធៀប ដើម្បីប្រៀបធៀប pointers ពីរ វាមានន័យតែពេលណា pointers ទាំងពីរបានចង្អុលទៅកាន់ធាតុនៃ array ដូចគ្នា។ ហើយលទ្ធផលដែលទទួលបានអាស្រ័យទៅលើទីតាំងធៀបនៃធាតុពីររបស់ array។ ឧទាហរណ៍ បន្ទាប់ពីកំណត់តម្លៃខាងក្រោម៖

```
p = &a[5];
q = &a[1];
```

តម្លៃរបស់ p <= q គឺ 0 (សូន្យ) ហើយតម្លៃរបស់ p >= q គឺ 1 (មួយ)។

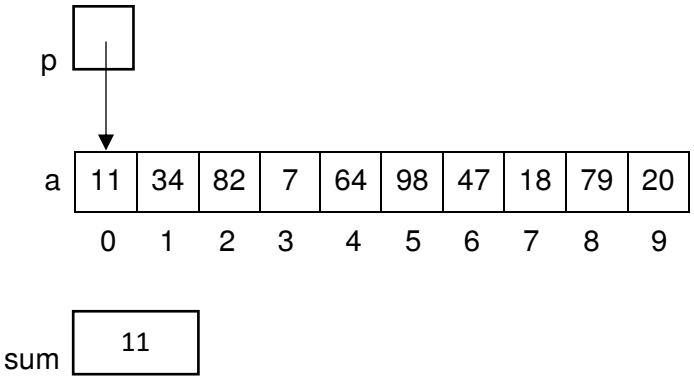
៨. **ការប្រើ** Pointer **សម្រាប់ដំណើរការ** array

ប្រមាណវិធី pointer អាចឲ្យយើងមើលធាតុរបស់ array ដោយបង្កើនមួយឯកតាទៅលើអញ្ញាត pointer ម្តងមួយតម្លៃ។។ code ខាងក្រោមនេះធ្វើផលបូកធាតុនៃ array a មួយនឹងបង្ហាញពីវិធីធ្វើ។ នៅក្នុងឧទាហរណ៍នេះ អញ្ញាត pointer p ចង្អុលទៅកាន់ a[0] ពេលចាប់ផ្តើមដំបូង។ រៀងរាល់ពេល loop ប្រតិបត្តិ, p ត្រូវកើនមួយតម្លៃឯកតា តាមលទ្ធផល វាចង្អុលទៅកាន់ a[1] បន្ទាប់មក a[2] និង -ល-។ ដំណើរការ loop បញ្ចប់ កាលណា p ដើរហួសធាតុចុងក្រោយរបស់ a។

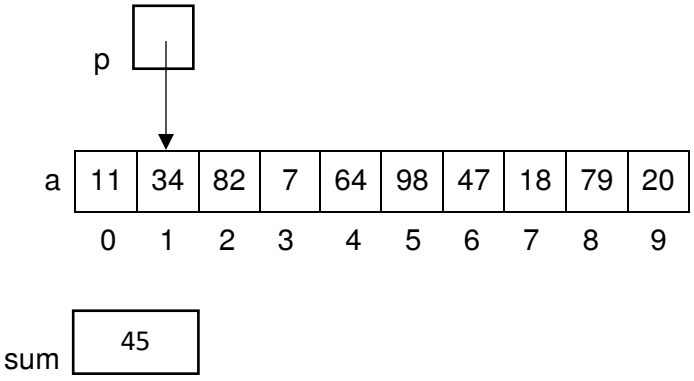
```
#define N 10
...
int a[N], sum, *p;
...
sum = 0;
for (p = &a[0];  p < &a[N]; p++)
   sum += *p;
```

រូបខាងក្រោមបង្ហាញតម្លៃរបស់ a, sum និង p នៅចុងបញ្ចប់នៃបីដំណើរវង្វល់ជុំដំបូង (មុននឹង p ត្រូវបង្កើនមួយឯកតា)។

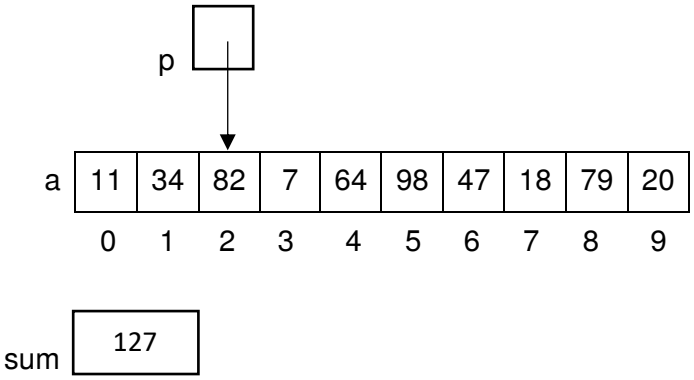
ជុំទី១



ជុំទី២



ជុំទី៣



លក្ខខណ្ឌ p < &a[N] នៅក្នុងឃ្លា for ត្រូវបានអធិប្បាយដោយឡែក។ ភាពចម្លែកនេះ វាហាក់ដូចជាប្រើសញ្ញាណនព្វន្តអស័យដ្ឋានត្រឹមត្រូវទៅកាន់ a[N] ទោះជាធាតុនេះមិនស្ថិតនៅក៏ដោយ (a មាន index ពី 0 ដល់ N – 1)។ ការប្រើ a[N] នៅក្នុងលក្ខណៈបែបនេះមានសុវត្ថិភាពខ្លាំង ដោយ loop

មិនព្យាយាមពិនិត្យតម្លៃរបស់វា។ តួរបស់ loop នឹងត្រូវប្រតិបត្តិដោយ p ស្មើទៅនឹង &a[0], &a[1], …, &a[N–1] ប៉ុន្តែនៅពេលដែល p ស្មើនឹង &a[N] នោះ loop នឹងត្រូវបញ្ចប់។

ការបន្សំសញ្ញាណនព្វន្ត * និង ++

អ្នកសរសេរកម្មវិធីភាសា C តែងតែបន្សំសញ្ញាណនព្វន្ត * (indirection) និង ++ ក្នុងឃ្លាដំណើរការជាមួយនឹងធាតុ array។ ចូរពិចារណាលើករណីផ្ទុកតម្លៃទៅក្នុងធាតុ array រួចហើយបន្តឈានទៅធាតុនៅបន្ទាប់។ ការប្រើជាមួយនឹង index របស់ array គេអាចសរសេរ ៖

```
a[i++] = j;
```

បើសិនជា p ចង្អុលទៅកាន់ធាតុមួយនៃ array នោះ ឃ្លាដែលត្រូវគ្នាគួរសរសេរ ៖

```
*p++ = j;
```

ដោយសារការសរសេរ ++ ដាក់នៅក្រោយ p វាមានអទិភាពជាង * ដូចនេះ compiler យល់ឃើញដូចជាការសរសេរ

```
*(p++) = j;
```

តម្លៃរបស់ p++ គឺ p។ ដោយសារយើងប្រើ ++ ដាក់ពីខាងក្រោយ p បានជាវាមិនបង្កើនតម្លៃមួយឯកតា ដរាបណាកន្សោមរបស់វាត្រូវបានរង្វាយតម្លៃ។ ដូចនេះ តម្លៃរបស់ *(p++) ត្រូវគ្នានឹង *p បានន័យថា object ដែល p ចង្អុលទៅ។

តាមពិត *p++ មិនត្រឹមតែជាការបន្សំរវាង * និង ++ ដែលមានលក្ខណៈត្រឹមត្រូវប៉ុណ្ណោះទេ។ ជាក់ស្តែង យើងអាចសរសេរ (*p)++ ដែលឲ្យតម្លៃរបស់ object ដែល p ចង្អុលទៅ ហើយបន្ទាប់មកបង្កើនតម្លៃឲ្យ object នោះ (p ខ្លួនវាមិនផ្លាស់ប្តូរឡើយ)។ បើអ្នកនៅតែមានការច្រឡំ ចូរសង្កេតឃ្លាក្នុងតារាងខាងក្រោមនេះ៖

កន្សោមលេខ	អត្ថន័យ
*p++ ឬ *(p++)	តម្លៃកន្សោម គឺ *p មុនកំណើន, កំណើនមួយឯកតាឲ្យ p ពេលក្រោយ
(*p)++	តម្លៃកន្សោម គឺ *p មុនកំណើន, កំណើនមួយឯកតាឲ្យ *p ពេលក្រោយ
*++p ឬ *(++p)	កំណើនមួយឯកតាឲ្យ p មុន, តម្លៃកន្សោម គឺ *p បន្ទាប់ពីកំណើនឯកតា
++*p ឬ ++(*p)	កំណើនមួយឯកតាឲ្យ *p មុន, តម្លៃកន្សោម គឺ *p បន្ទាប់ពីកំណើនឯកតា

ឃ្លាបន្សំទាំងបួនយ៉ាងខាងលើបានសរសេរនៅក្នុងកម្មវិធី ទោះជាមានឃ្លាបន្សំខ្លះគេនិយមសរសេរជាងក៏ដោយ។ យើងនឹងលើកយកឃ្លាមួយដែលមានការប្រើច្រើននៅក្នុង loop គឺ *p++ ដែលជំនួសឲ្យឃ្លាខាងក្រោម៖

```
for (p = &a[0] ; p < &a[N]; p++)
    sum += *p;
```

ដើម្បីបូកធាតុរបស់ array a យើងអាចសរសេរជា

```
p = &a[0];
while (p < &a[N])
   sum += *p++;
```

ការបន្សំសញ្ញាណនព្វន្ត * និង -- គឺដូចទៅនឹងសញ្ញាណនព្វន្ត * និង ++ ដែរ។

៩. ការប្រើឈ្មោះ array ជា pointer

ប្រមាណវិធី pointer គឺជាមធ្យោបាយមួយដែល arrays និង pointers មានទំនាក់ទំនងគ្នា ប៉ុន្តែវាមិនគ្រាន់តែមានទំនាក់ទំនងគ្នាទេ។ នេះជាទំនាក់ទំនងសំខាន់មួយទៀត : ឈ្មោះរបស់ array មួយ អាចត្រូវបានប្រើដូចជា pointer មួយចង្អុលទៅកាន់ធាតុទីមួយនៃ array។ ការទំនាក់ទំនងនេះសម្រួលប្រមាណវិធី pointer និងធ្វើឲ្យទាំង arrays និង pointers មានការប្រើច្រើនទៀត។

ឧទាហរណ៍ ឧបមាថា a ត្រូវបានប្រកាសដូចខាងក្រោម:

```
int a[10];
```

ការប្រើ a ដូចជា pointer មួយចង្អុលទៅកាន់ធាតុទីមួយក្នុង array នោះ យើងអាចកែប្រែ a[0] :

```
*a = 7;          /* stores 7 in a[0]  */
```

យើងអាចកែប្រែ a[1] តាមរយៈ pointer a+1 :

```
*(a+1) = 12;    /* stores 12 in a[1]  */
```

ជាទូទៅ a + i គឺដូចគ្នាទៅនឹង &a[i] (ទាំងពីរនេះតាងឲ្យ pointer មួយចង្អុលទៅកាន់ធាតុ i របស់ a) ហើយ *(a+i) គឺសមមូលទៅនឹង a[i] (ទាំងពីរនេះតាងឲ្យធាតុ i មួយខ្លួនវា)។ គេអាចនិយាយ ម៉្យាងទៀតថា index របស់ array អាចមើលឃើញបានដូចទៅនឹងទម្រង់មួយនៃប្រមាណវិធី pointer។

តាមពិត ឈ្មោះ array មួយអាចប្រើជា pointer មួយបង្កឲ្យវាងាយសរសេរ loop ជាជំហានៗតាមរយៈ array មួយ។ ចូរពិចារណា loop ខាងក្រោម៖

```
for (p=&a[0]; p < &a[N]; p++)
   sum += *p;
```

ដើម្បីធ្វើឲ្យ loop មានទម្រង់ងាយ គេអាចជំនួស &a[0] ដោយ a និង &a[N] ដោយ a + N :

```
for (p = a; p < a + N; a++)
   sum += *p;
```

ទោះជា ឈ្មោះ array មួយអាចប្រើជា pointer បាន វាមិនអាចកំណត់តម្លៃថ្មីឡើយ។ ការព្យាយាមធ្វើឲ្យវាចង្អុលទៅកន្លែងណាមួយនោះនាំឲ្យ error មួយ :

```
while (*a != 0)
   a++;          /*** WRONG ***/
```

នោះមិនមានការខាតបង់ច្រើនឡើយ ជានិច្ចជាកាលយើងអាចចម្លង a ទៅលើអញ្ញាត pointer ពេលនោះប្តូរអញ្ញាត pointer :

```
p = a;
while (*p != 0)
   p++;
```

កម្មវិធីខាងក្រោមនេះទទួលតម្លៃជាចំនួនគត់ផ្ទុកក្នុង array មួយចំនួន N តម្លៃ។ ក្រោយពីបញ្ចូលតម្លៃទាំងអស់ទៅក្នុង array ហើយនោះវាបង្ហាញតម្លៃទាំងនោះបញ្ច្រាសមកលើអេក្រង់។

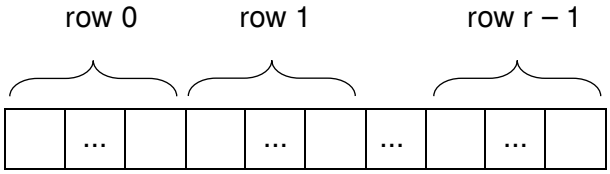
```
/* Reverses a series of numbers (pointer version) */
#include <stdio.h>
#include <conio.h>
#define N 10
void main(){
  int a[N], *p;  // ការប្រកាសអញ្ញាត array a ចំនួន N ធាតុនិង អញ្ញាត *p
  printf("Enter %d numbers : ", N);  // បង្ហាញតម្លៃរបស់ N មកលើអេក្រង់
  for (p = a; p < a+N; p++)
     scanf("%d", p);   // ទទួលតម្លៃពី keyboard និងផ្ទុកឲ្យអញ្ញាត pointer p
  printf("In reverse order : ");  // បង្ហាញឃ្លានេះមកលើអេក្រង់
  for (p = a+N-1; p >= a; p--)
     printf(" %d", *p); // បង្ហាញតម្លៃរបស់ *p មកលើអេក្រង់
  printf("\n");
  getch();
```

}

១០. ការប្រើ Pointer និង array ច្រើនវិមាត្រ

Pointers មិនគ្រាន់តែអាចចង្អុលធាតុនៃ array មួយវិមាត្រប៉ុណ្ណោះទេ តែវាអាចចង្អុលទៅលើធាតុនៃ array ច្រើនវិមាត្រផងដែរ។ នៅក្នុងផ្នែកនេះ យើងនឹងបង្ហាញពីវិធីប្រើ pointers សម្រាប់ដំណើរការធាតុនៃ array ច្រើនវិមាត្រ។ ដើម្បីសម្រួលដល់ការបង្ហាញនេះ យើងលើកយកពីការប្រើ pointers ជាមួយនឹង array ពីរវិមាត្រ ដែលវាដូចទៅនឹងការប្រើ pointers ជាមួយនឹង array ច្រើនវិមាត្រដទៃទៀតដែរ។

យើងបានឃើញហើយចំពោះ array ពីរវិមាត្រកាលពីជំពូកមុន។ ក្នុងភាសា C, វាផ្ទុកតម្លៃក្នុង array ពីរវិមាត្រតាមជួរដេក (row) ទី0 មុនគេ បន្ទាប់មកផ្ទុកតាមជួរដេកទី១ ហើយនិងជួរដេកជាបន្តបន្ទាប់។ array មួយមានជួរដេក r ដែលមានរូបភាពដូចខាងក្រោម:



យើងអាចយកប្រយោជន៍នៃការរៀបនេះនៅពេលធ្វើការជាមួយ pointers។ បើយើងបង្កើត pointer p មួយចង្អុលទៅកាន់ធាតុទីមួយក្នុង array ពីរវិមាត្រ (ពោលគឺ row ទី 0 និង column ទី 0) យើងអាចប្រើគ្រប់ធាតុក្នុង array ដោយបង្កើនតម្លៃ p ដដែលៗ។

ឧទាហរណ៍ ចូរសង្កេតបញ្ហានៃការកំណត់តម្លៃដំបូងគ្រប់ធាតុទាំងអស់នៃ array ពីរវិមាត្រទៅនឹងតម្លៃសូន្យ។ ឧបមាថា array ត្រូវបានប្រកាសដូចខាងក្រោម៖

```
int a[NUM_ROWS][NUM_COLS];
```

វិធីខាងក្រោមនេះត្រូវប្រើ loop មួយនៅក្នុង loop មួយទៀត៖

```
int row, col;
...
for (row = 0; row < NUM_ROWS; row++)
   for (col = 0; col < NUM_COLS; col++)
     a[row][col] = 0;
```

ក៏ប៉ុន្តែយើងឃើញ a ដូច array នៃចំនួនគត់មួយវិមាត្រ (មើលពីរបៀបផ្ទុកតម្លៃ) យើងអាចជំនួស loop មួយគូដោយ loop តែមួយ:

```
int *p;
...
for (p = &a[0][0]; p <= &a[NUM_ROWS – 1][NUM_COLS – 1]; p++)
    *p = 0;
```

loop ចាប់ផ្តើមជាមួយនឹង p ដែលចង្អុលទៅកាន់ a[0][0]។ បង្កើនតម្លៃ p មុន វាចង្អុលទៅកាន់ a[0][1], a[0][2], a[0][3] - ល -។ នៅពេល p ទៅដល់ a[0][NUM_COLS – 1] (ធាតុចុងក្រោយក្នុង row 0), បង្កើនតម្លៃម្តងទៀត ធ្វើឲ្យ p ចង្អុលទៅកាន់ a[1][0], ធាតុទី 1 ក្នុង row 1។ ដំណើរការបន្តរហូតដល់ p ទៅដល់ a[NUM_ROWS – 1][NUM_COLS – 1] ដែលជាធាតុចុងក្រោយរបស់ array។

១០.១ ដំណើរការជាមួយនឹង row របស់ array ច្រើនវិមាត្រ

តើមានអ្វីដែលអាចដំណើរការធាតុទាំងអស់តាមជួរដេក (row) របស់ array ច្រើនវិមាត្រដែរឬទេ?

យើងមានជម្រើសនៃការប្រើអញ្ញាត pointer p។ ដើម្បីចូលទៅប្រើធាតុរបស់ row i, យើងគួរកំណត់តម្លៃដំបូងឲ្យ p ដោយចង្អុលទៅធាតុ 0 តាមជួរដេក row i ក្នុង array a :

```
p = &a[i][0];
```

ឬ យើងអាចសរសេរទម្រង់ងាយ

```
p = a[i];
```

ចំពោះ array a ពីរវិមាត្រណាក៏ដោយ កន្សោម a[i] គឺជា pointer មួយនៅត្រង់ធាតុទី ០ តាមជួរដេក row i។ ដើម្បីមើលឃើញពីលក្ខណៈនេះធ្វើការ យើងប្រើរូបមន្តមួយដែលទាក់ទង index របស់ array ទៅនឹងប្រមាណវិធី pointer : ចំពោះ array a ណាមួយក៏ដោយ កន្សោម a[i] គឺស្មើនឹង *(a+i)។ ហេតុនេះ &a[i][0] គឺដូចគ្នាទៅនឹង &(*(a[i] +0)), ដែលវាស្មើនឹង &*a[i] ក៏ដូចនឹង a[i] ដោយសារសញ្ញាណនព្វន្ត & និង * ទុកជាមោយ:។ យើងនឹងប្រើការសម្រួលនេះនៅក្នុង loop ខាងក្រោមដែលលុបតម្លៃ row i របស់ array a :

```
int a[NUM_ROWS][NUM_COLS], *p, i;
...
```

```
for (p = a[i]; p < a[i] + NUM_COLS; p++)
      *p = 0;
```

ដោយ a[i] គឺជា pointer មួយទៅកាន់ row i របស់ array a, យើងអាចបញ្ជូន a[i] ទៅកាន់អនុគមន៍ដែលរំពឹងគិតទុក array មួយវិមាត្រ ជា argument។ គេអាចនិយាយម៉្យាងទៀត អនុគមន៍មួយរៀបចំឡើងដើម្បីធ្វើការជាមួយនឹង array មួយវិមាត្រ ហើយក៏ធ្វើការជាមួយនឹង row ផងដែរដែលជារបស់ array ពីរវិមាត្រ។

១០.២ ដំណើរការជាមួយនឹង Columns របស់ array ច្រើនវិមាត្រ

ដំណើរការធាតុទាំងអស់តាមជួរឈររបស់ array ពីរវិមាត្រមិនមែនមានលក្ខណៈងាយស្រួលឡើយ ពីព្រោះ array ត្រូវបានផ្ទុកតាមជួរដេក មិនមែនតាមជួរឈរឡើយ។ នេះជា loop ដែលលុបតម្លៃតាមជួរឈរ column i របស់ array a :

```
int a[NUM_ROWS][NUM_COLS], (*p)[NUM_COLS], i;
...
for (p = &a[0]; p < &a[NUM_ROWS]; p++)
      (*p)[i] = 0;
```

គេបានប្រកាស p ជា pointer មួយទៅលើ array មួយប្រវែង NUM_COLS ដែលធាតុរបស់វា ជាចំនួនគត់។ សញ្ញាវង់ក្រចកជុំវិញ *p ជា (*p)[NUM_COLS] ដែលជាតម្រូវការ ហើយបើគ្មានវាទេនោះ compiler នឹងចាត់ទុក p ដូចទៅនឹង arrays នៃ pointers ជំនួសឲ្យ pointer ចង្អុលទៅកាន់ array។ កន្សោម p++ ពីមុខ p ដើម្បីចាប់ផ្តើម row នៅបន្ទាប់។ នៅក្នុងកន្សោម (*p)[i], *p តាងឲ្យ row ទាំងមូលរបស់ a, ដូចនេះ (*p)[i] ជ្រើសយកធាតុក្នុង column i របស់ row នោះ។ សញ្ញាវង់ក្រចក ក្នុង (*p)[i] គឺមានសារៈសំខាន់ណាស់ ពីព្រោះ compiler នឹងបកប្រែ *p[i] ដូចទៅនឹង *(p[i])។

១១. ការប្រើ pointer និង character strings

Pointer អាចត្រូវបានគេប្រើសម្រាប់បង្កើត strings ផងដែរ។ អញ្ញាត pointer នៃប្រភេទទិន្នន័យ char ត្រូវបានចាត់ទុកដូចជា string។

```
char *str = "Hello";
```

ឃ្លានេះបង្កើតនូវ string មួយនិងផ្ទុកអស័យដ្ឋានរបស់វានៅក្នុងអញ្ញាត pointer str។ ពេលនេះ pointer str ចង្អុលទៅកាន់តួអក្សរទី១នៃ string "Hello"។ រឿងសំខាន់មួយទៀតដែលគួរកត់សម្គាល់នោះគឺ string បង្កើតឡើងដោយប្រើ pointer char ដែលអាចកំណត់តម្លៃនៅ runtime។

```
char *str;
str = "Hello";   //thi is Legal
```

តម្លៃរបស់ string អាចបង្ហាញមកលើអេក្រង់ដោយប្រើ printf() និង puts()។

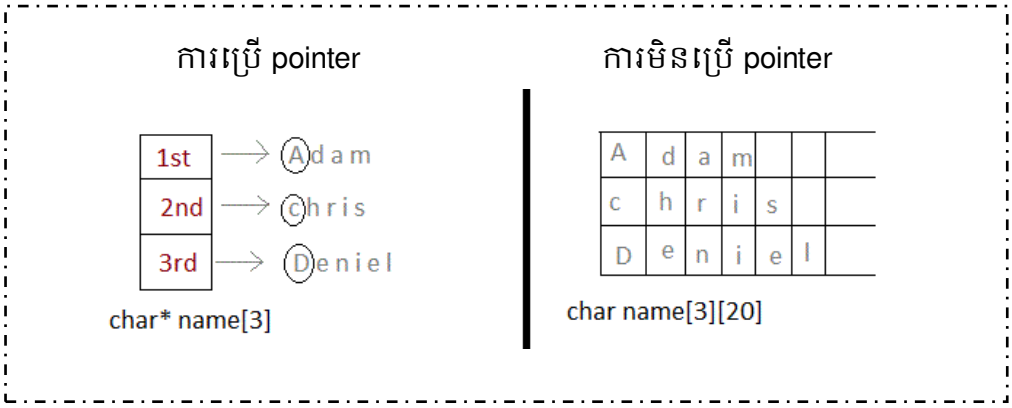
```
printf("%s", str);
puts(str);
```

គួរកត់សម្គាល់ថា str គឺជា pointer មួយដែលចង្អុលទៅកាន់ string ហើយវាក៏ជាឈ្មោះរបស់ string។ ហេតុនេះ យើងមិនចាំបាច់ប្រើសញ្ញាណនព្វន្ត * ឡើយ។

១២. អំពី array នៃ pointers

យើងមាន array នៃ pointers ផងដែរ។ pointers មានសារប្រយោជន៍ណាស់ក្នុងការប្រើជា មួយ array ប្រភេទ character ជាមួយនឹង rows ដែលមានប្រវែងផ្សេងគ្នា។

```
char *name[3]={
          "Adam",
          "chris",
          "Deniel"
           };
// Now see same array without using pointer
char name[3][20]= {
          "Adam",
          "chris",
          "Deniel"
           };
```



១៣. អំពី null pointer

pointer មួយដែលមិនបានកំណត់តម្លៃណាមួយ ក៏ប៉ុន្តែ NULL ត្រូវបានគេស្គាល់ថាជា NULL pointer។ លក្ខណៈនេះធ្វើបាននៅពេលប្រកាសអញ្ញាត។ NULL pointer គឺជាចំនួនថេរមួយដែលត្រូវកំណត់ក្នុង Libraries ស្តង់ដារ ដោយតម្លៃសូន្យ។

```
#include <stdio.h>
#include <conio.h>
void main(){
  int *ptr = NULL;
  printf("Value of ptr is: %x", ptr);
  getch();
}
```

លទ្ធផលទទួលបាន៖ Value of ptr is: 0

១៤. អំពី dynamic storage allocation

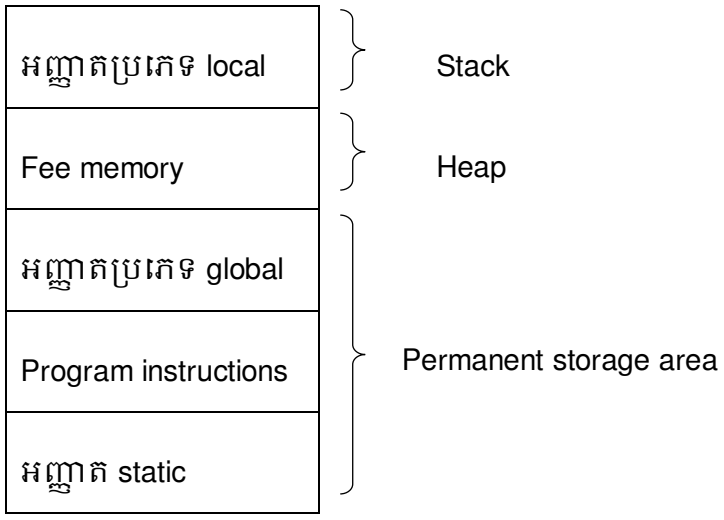
ដំណើរការនៃការដាក់ទីតាំង memory នៅពេល runtime ត្រូវបានគេស្គាល់ថា dynamic memory allocation។ ដំណើរ Library ត្រូវបានទទួលស្គាល់ថា "memory management functions" ត្រូវបានប្រើ

សម្រាប់ដាក់ទីតាំង memory និង បំបាត់ចេញ memory ក្នុងពេលប្រតិបត្តិការរបស់កម្មវិធី។ អនុគមន៍ទាំងនេះត្រូវបានកំណត់នៅក្នុង stdlib.h។

អនុគមន៍	សេចក្តីអធិប្បាយ
malloc()	ដាក់ទីតាំងនូវទំហំគិតជា bytes តាមសំណើ ហើយឲ្យតម្លៃនូវ void pointer ដែលចង្អុលទៅកាន់ byte ទីមួយនៃទីតាំង memory។
calloc()	ការដាក់ទីតាំងក្នុង memory សម្រាប់ធាតុនៃ array មួយ ហើយកំណត់តម្លៃដំបូងទៅឲ្យវាស្មើ 0 រួចហើយឲ្យតម្លៃជា void pointer ដែលចង្អុលទៅ memory។
free()	បញ្ចេញពីទីតាំង memory នូវអ្វីដែលមានពីមុន
realloc()	កែប្រែតម្លៃទំហំនៃទីតាំង memory មានពីមុន

១៤.១ ដំណើរការដាក់ក្នុងទីតាំង memory (Memory allocation process)

អញ្ញាតប្រភេទ global, អញ្ញាត static និង ពាក្យបញ្ជាកម្មវិធី (program instructions) យក memory ក្នុងទីតាំងផ្ទុកជាអចិន្ត្រៃយ៍ រីឯអញ្ញាតប្រភេទ local ត្រូវបានផ្ទុកក្នុងទីតាំងមួយហៅថា Stack។ ទំហំ memory ចន្លោះទីតាំងទាំងពីរត្រូវបានគេហៅថា ទីតាំង Heap។ ផ្នែកនេះត្រូវបានប្រើសម្រាប់ dynamic memory allocation ក្នុងពេលដំណើរការកម្មវិធី។ ទំហំទីតាំង Heap តែងប្រែប្រួល។



១៤.២ ការដាក់ក្នុងទីតាំង block នៃ memory

- អនុគមន៍ malloc() ត្រូវបានប្រើសម្រាប់ដាក់ក្នុងទីតាំង block នៃ memory នៅពេលដំណើរការ។ អនុគមន៍នេះបម្រុងទុកទីតាំង block នៃ memory របស់ទំហំដែលគេឲ្យ ហើយឲ្យតម្លៃជា pointer ប្រភេទ void។ នេះបានន័យថា យើងអាចកំណត់តម្លៃទៅឲ្យ pointer ប្រភេទទិន្នន័យណាមួយដោយប្រើ typecasting។ បើសិនវាដាក់ទៅទីតាំងមិនមានទំហំគ្រប់គ្រាន់នោះ វានឹងឲ្យតម្លៃជា NULL pointer។ ទម្រង់ទូទៅរបស់គឺ ៖

```
void *malloc(size_t size);
```

ឧទាហរណ៍នៃការប្រើ malloc() ៖

```
#include <stdio.h>
#include <cstring.h>
#include <conio.h>
void main(){
     char *mem_allocation;
     /* memory is allocated dynamically */
     mem_allocation = (char*) malloc(20 * sizeof(char));
     if( mem_allocation== NULL ){
        printf("Couldn't able to allocate requested
              memory\n");
     } else {
        strcpy( mem_allocation, "Hello, world!");
     }
     printf("Dynamically allocated memory content : \
             %s\n", mem_allocation );
     free(mem_allocation);
     getch();
}
```

- អនុគមន៍ calloc() គឺជាអនុគមន៍សម្រាប់ដាក់ក្នុងទីតាំង memory មួយទៀតដែលវាប្រើនៅពេលដំណើរការកម្មវិធី។ អនុគមន៍ calloc() នេះជាធម្មតាប្រើសម្រាប់ដាក់ក្នុង memory ទៅឲ្យប្រភេទទិន្នន័យដល់ arrays និង structures។ បើសិនវាដាក់ទៅទីតាំងមិនមានទំហំគ្រប់គ្រាន់នោះ វានឹងឲ្យតម្លៃជា NULL pointer។ ទម្រង់ទូទៅរបស់វាគឺ ៖

void *calloc(size_t nelements, size_t elementSize);

ឧទាហរណ៍នៃការប្រើអនុគមន៍ calloc() ៖

```
#include <stdio.h>
#include <cstring.h>
#include <conio.h>
void main() {
      char *mem_allocation;
      /* memory is allocated dynamically */
      mem_allocation = (char *)calloc(20, sizeof(char));
      if(mem_allocation== NULL ) {
         printf("Couldn't able to allocate requested
              memory\n");
      } else {
          strcpy( mem_allocation, "Hello, world!");
      }
      printf("Dynamically allocated memory content : \
             %s\n", mem_allocation );
      free(mem_allocation);
      getch();
}
```

- អនុគមន៍ realloc() ធ្វើការប្តូរទំហំ memory ដែលបានបង្កើតទីតាំង memory ដោយប្រើ malloc()។ វាទទួលយក pointer ជាប៉ារ៉ាម៉ែត្រដែលចង្អុលទៅកាន់ទីតាំង memory និងទំហំថ្មីដែលត្រូវការ។ បើទំហំថយចុះ នោះទិន្នន័យអាចបាត់បង់។ បើទំហំកើនឡើង នោះអនុគមន៍មិនអាចពង្រីកទីតាំងដែលមានរួចហើយ វានឹងបង្កើតទីតាំងថ្មីក្នុង memory ហើយចម្លងទិន្នន័យចូល។

ទម្រង់ទូទៅរបស់វាគឺ ៖

void *realloc(void *pointer, size_t size);

ឧទាហរណ៍នៃការប្រើ realloc() ៖

```
#include <stdio.h>
#include <cstring.h>
#include <conio.h>
void main(){
    char *mem_allocation;
    /* memory is allocated dynamically */
    mem_allocation = (char *) malloc(20 * sizeof(char));
    if( mem_allocation == NULL ) {
        printf("Couldn't able to allocate requested
             memory\n");
    } else {
       strcpy(mem_allocation, "Hello, world!");
    }
    printf("Dynamically allocated memory content : %s\n",
          mem_allocation );
    mem_allocation= (char *) realloc(mem_allocation,
                                 100 * sizeof(char));
    if(mem_allocation == NULL) {
        printf("Couldn't able to allocate requested
             memory\n");
    } else {
        strcpy(mem_allocation,"space is extended upto
            100 characters");
    }
    printf("Resized memory : %s\n", mem_allocation);
    free(mem_allocation);
    getch();
}
```

១៤.៣ ភាពខុសគ្នារវាង malloc( ) និង calloc( )

calloc()	malloc()

calloc() កំណត់តម្លៃដំបូងឲ្យទីតាំង memory ដោយតម្លៃ 0	malloc() កំណត់តម្លៃដំបូងឲ្យទីតាំង memory ដោយតម្លៃដកជាប់ក្នុង memory
ចំនួន argument គឺ 2	ចំនួន argument គឺ 1
Syntax : (cast_type *) calloc(blocks , size_of_block);	Syntax : (cast_type *) malloc(Size_in_bytes);

១៥. អំពី memory storage deallocation : free( )

Memory ដែលត្រូវបានបង្កើតទីតាំងដោយប្រើ malloc(), realloc() ឬ calloc() នោះត្រូវបំបាត់ចេញពីកន្លែងផ្ទុកនៃ memory កាលណាគេលែងត្រូវការ។ លក្ខណៈនេះគេធ្វើឡើងជារឿយៗ ដើម្បីជៀសវាងការប្រើទីតាំង memory កាន់តែច្រើនឡើងៗដែលអាចនាំឲ្យគេទទួលបានលទ្ធផលមិនសម្រេចក្នុងការបង្កើតទីតាំង memory ជាយថាហេតុណាមួយ។ ទោះជាយ៉ាងណាក៏ដោយ memory មិនត្រូវបានបំបាត់ចេញនូវទីតាំងផ្ទុកដោយប្រើ free() កាលណាកម្មវិធីកំពុងប្រើត្រូវបញ្ឈប់ដោយ ប្រព័ន្ធប្រតិបត្តិការ។ ទម្រង់ទូទៅរបស់វា គឺ៖

```
void free(void *pointer);
```

code ខាងក្រោមបង្ហាញពីការប្រើអនុគមន៍ free() :

```
int *myStuff = malloc( 20 * sizeof(int));
if (myStuff != NULL){
   /* more statements here */
   /* time to release myStuff */
   free( myStuff );
}
```

ក្រៅពីនេះ យើងមានឧទាហរណ៍មួយទៀតដែលបង្ហាញពីការប្រើអនុគមន៍ free() ក្នុងនេះគេបានប្រកាស array មួយនិងកំណត់តម្លៃទៅឲ្យធាតុទី៤នៃ array :

```
int my_array[10];
my_array[3] = 99;
```

code ខាងក្រោមនេះមានលក្ខណៈដូចគ្នាទៅនឹង code ខាងលើដែរ តែវាប្រើទីតាំង memory :

```
int *pointer;
pointer = malloc(10 * sizeof(int));
*(pointer+3) = 99;
```

ទម្រង់ទាញយកតម្លៃពី pointer មានការពិបាកអាន ដូចនេះ ទម្រង់នៃការប្រើតាម array ធម្មតា អាចត្រូវបានគេប្រើគ្នា គឺ [និង] ដូចទៅនឹងសញ្ញាណខ្លួន ៖

```
pointer[3] = 99;
```

កាលណា array មិនត្រូវការប្រើទៀតនោះ ទីតាំង memory ត្រូវបំបាត់ចេញដោយសរសេរ ៖

```
free(pointer);
pointer = NULL;
```

ការកំណត់តម្លៃ NULL ទៅឲ្យ pointer មិនមែនជាការចាំបាច់នោះឡើយ ក៏ប៉ុន្តែវាជាការអនុវត្ត ដ៏ល្អដូចដែលវាអាចបណ្តាលឲ្យមាន error កើតឡើងនៅពេលដែល pointer មួយបានប្រើបន្ទាប់ពី memory ត្រូវលុបបំបាត់ទីតាំងប្រើនោះ។

សំណួរឆ្លងលំហាត់

១. អ្វីទៅហៅថា pointer ? តើ pointer ប្រើសម្រាប់ធ្វើអ្វី?

២. ចូរនិយាយពីភាពខុសគ្នារវាង calloc() និង malloc()។

៣. តើពេលណាគេប្រើអនុគមន៍ free() ?

៤. ចំពោះកម្មវិធីខាងក្រោមនេះ តើគេនឹងទទួលបានលទ្ធផលអ្វី?

```
#include <stdio.h>
#include <conio.h>
void main(){
    int arr[2][2][2] = {10, 2, 3, 4, 5, 6, 7, 8};
    int *p, *q;
    p = &arr[1][1][1];
    q = (int*) arr;
    printf("%d, %d\n", *p, *q);
    getch();
}
```

៥. ចំពោះកម្មវិធីខាងក្រោមនេះ តើគេនឹងទទួលបានលទ្ធផលអ្វី?

```
#include <stdio.h>
#include <conio.h>
```

```
void main(){
   static char *s[] = {"black", "white", "pink", "violet"};
   char **ptr[] = {s+3, s+2, s+1, s}, ***p;
   p = ptr;
   ++p;
   printf("%s", **p+1);
   getch();
}
```

៦. ចំពោះកម្មវិធីខាងក្រោមនេះ តើគេនឹងទទួលបានលទ្ធផលអ្វី?

```
#include <stdio.h>
#include <conio.h>
void main(){
    int i=3, *j;
    j = &i;
    printf("%d\n", i**j*i+*j);
    getch();
}
```

៧. ចំពោះកម្មវិធីខាងក្រោមនេះ តើគេនឹងទទួលបានលទ្ធផលអ្វី?

```
#include <stdio.h>
#include <conio.h>
void change(int *b, int n) {
    int i;
    for(i=0; i<n; i++)
      *(b+1) = *(b+i)+5;
}
void main(){
    int i, a[] = {2, 4, 6, 8, 10};
    change(a, 5);
```

```
    for(i=0; i<=4; i++)
        printf("%d, ", a[i]);
    getch();
}
```