Sistemes Operatius 1

Sessió de problemes 9 – Maig del 2022

Punters a C

Els punters són un tipus de variable característics del llenguatge C. Al llenguatge C, determinades tasques, com la gestió de la memòria dinàmica, no es pot realitzar sense l'ús de punters. A més, poder programar amb punters permet realitzar certes operacions de forma més eficient. Aquesta sessió se centra en l'ús de punters ja que en la pràctica 4 es faran servir i és molt convenient entendre els concepte associat.

Què és un punter? Un punter és, intuïtivament, un sencer sense signe que apunta a una determinada direcció de memòria. En comptes de manipular les variables directament, la idea és manipular direccions de memòria escrivint o llegint de/a la direcció de memòria.

Els exemples que es mostren a continuació s'han obtingut de la següent adreça web:
`https://www.tutorialspoint.com/cprogramming/c_pointers.htm`

És habitual utitilitzar punters amb tipus, per exemple,

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

La variable `ip`, per exemple, és una variable que apunta a una direcció de memòria en què s'emmagatzema un sencer.

Exemple 1:

Vegem un exemple de com podem modificar el valor d'un sencer mitjançant un punter:

```
int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var);
   printf("Address stored in ip variable: %x\n", ip);
   printf("Value of *ip variable: %d\n", *ip);

   *ip = 30; /* Change value of the contents of *ip */
   printf("Value of variable var: %d\n", var);

   return 0;
}
```

Els punters tenen múltiples conceptes associats que són importants en el llenguatge C. Entendre'ls permet fer, com s'ha comentat abans, determinades operacions de forma més eficient. Aquí només ens centrarem en els conceptes que són necessaris per a fer la pràctica 4. Per a la resta de conceptes mirar l'enllaç esmentat abans.

Exemple 2:

Una de les operacions habituals per a la qual es fan servir els punters són les manipulacions de vectors. Aquí tenim un exemple que en mostra el concepte:

```
const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have array address in pointer */
   ptr = var;

   for ( i = 0; i < MAX; i++) {

      printf("Address of var[%d] = %x\n", i, &var[i]);
      printf("Value of ptr for iteration %d is %d\n", i, ptr);

      /* move to the next location */
      ptr++;
   }

   return 0;
}
```

Aquest exemple vol mostrar un dels conceptes interessants en la manipulació de vectors: podem accedir als valor d'un vector fent servir punters (`ptr`) i no pas la variable original (`var`). Ho podem fer ja que es poden realitzar operacions aritmètiques (`++`, `--`, `+`, `-`) amb els punters.

1. Què fa `ptr++`? En quant s'incrementa el valor de `ptr` cada cop que fem `ptr++`? Per què?
2. Modifica l'aplicació anterior perquè els valors del vector `var` s'incrementin en 1 fent servir els punters `ptr`.

Exemple 3:

Un dels usos habituals dels punters és a la memòria dinàmica. En particular, la funció malloc permet demanar, de forma dinàmica, al sistema operatiu el nombre de bytes que li fan falta al programa en aquell moment.

```
int main () {

   int  i, *var;

   var = malloc(1000 * sizeof(int));

   for ( i = 0; i < 1000; i++) {
       var[i] = 2 * i + 1;
   }

   free(var);

   return 0;
}
```

Es demana

1. Quants bytes es demanen al sistema operatiu?
2. Modifiqueu el programa perquè els valors del vector s'inicialitzin (als valors que es veuen al codi) fent servir un punter `ptr` en comptes d'inicialitzar-los fent servir `var[i]`.
3. Per què creieu que és més eficient modificar els valor de la variale `var` fent servir punters en comptes d'accedir-hi directament fent servir `var[i]`? En altres paraules, que creieu que fa "internament" el codi fer per accedir a `var[i]`?

Altres operacions que són habituals de realitzar són el pas de punters a funcions. Això permet que una funció (per exemple, la funció main) passi a una altra funció el valor de `var`. Dintre d'aquesta funció es poden modificar els valors del vector. Aquestes modificacions es podran "veure" en retornar de la funció.

```
void funcio(int *ptr)
{
   int  i;
   for(i = 0; i < 1000; i++)
       ptr[i] = 2 * i + 1;
}

int main () {
   int  *var;

   var = malloc(1000 * sizeof(int));
   funcio(var);
   free(var);

   return 0;
}
```

De la mateixa forma que es poden passar punters a funcions, també es poden retornar punters de funcions.

Exemple 4:

Un darrer tipus de punter que volem presentar són els punters sense tipus associat (int, float, double, ...). Aquests tipus de punters s'anomenen **punters genèrics** i seran els tipus de punters amb què treballareu a la pràctica 4. Per tant, és important que us quedi clar aquest tipus de punter.

```
int main () {
   void  *ptr;

   ptr = malloc(4000);

   // blah blah blah

   free(ptr);

   return 0;
}
```

En aquest exemple només estem indicant que volem reservar 4000 bytes de memòria, però no estem dient (en declarar la variable) quin tipus d'informació (int, float, ...) hi emmagatzemarem.

Com hi podem emmagatzemar informació? Ara tocar manipular punters...

```
int main () {
   int *p_int;
   void  *ptr;

   ptr = malloc(4000);

   p_int = (int *) (ptr + 3);
   *p_int = 50;

   printf("Valor de ptr: %x\n", ptr);
   printf("Valor de p_int: %x\n", p_int);

   free(ptr);

   return 0;
}
```

Fixeu-vos que en aquest codi és declara un punter de tipus genèric (`ptr`) així com un de tipus sencer (`p_int`). Després és fa una operació aritmètica amb el punter genèric, `ptr+3`, i s'assigna al punter de tipus sencer mitjançant un *casting*.

Què estem fent amb aquesta operació de suma de punters? En quan s'incrementa `ptr` en fer l'operació `ptr+3`?

On estem emmagatzemant el valor sencer? Analitzeu bé els valors que s'imprimeixen de `ptr` i `p_int`.

Exemple 5:

Finalment mostrem un exemple en què es mostra una de les formes que es pot fer servir per reservar memòria dinàmica per a una matriu. Reservar memòria per a una matriu de forma estàtica és ben senzill...

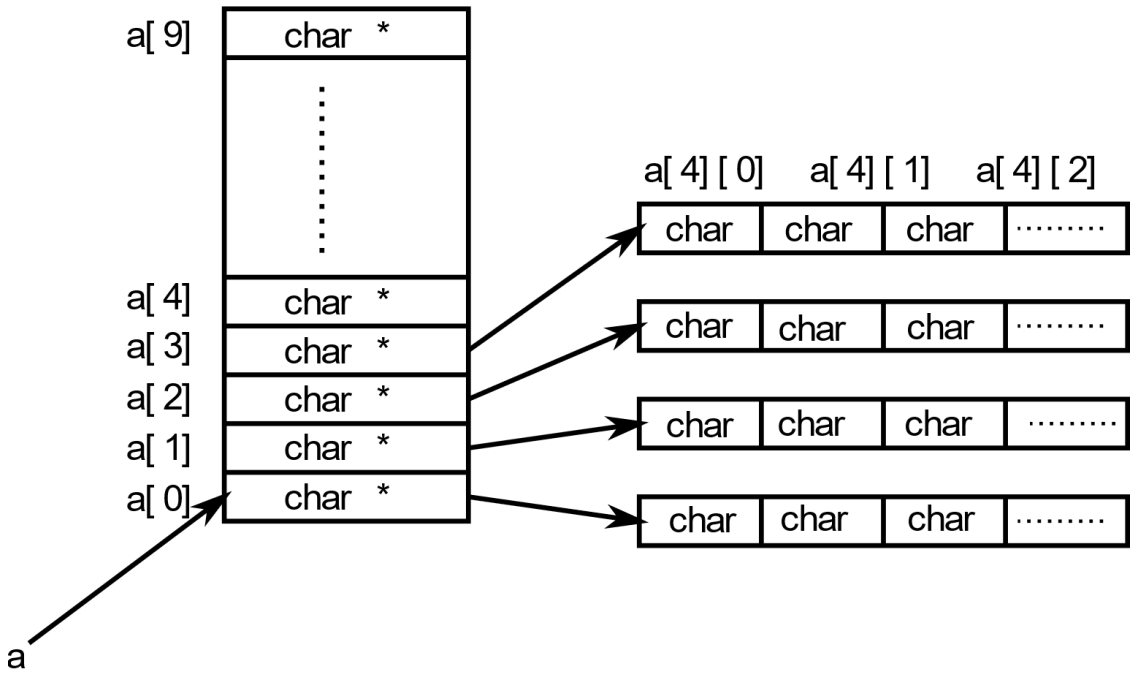
```
int main(void)
{
    char a[10][10]; // matriu de 10x10 de char

    a[0][0] = 0;
    a[0][0] = a[0][0] + 1;

    return 0;
}
```

El fet de reservar memòria de forma estàtica té els seus inconvenients. A la següent sessió de problemes veurem que el fet de fer-ho de forma estàtica no permet reservar matrius de mida gaire gran (més de 10MBytes, per exemple). És més convenient reservar la memòria de forma dinàmica.

En el llenguatge C, reservar memòria per a una matriu de forma dinàmica, implica un procediment una mica tediós. El primer cop que es fa pot semblar complexe. Un cop s'ha entès se'n pot treure profit a l'hora d'implementar determinades funcionalitats. A continuació es mostra com es pot reservar memòria dinàmica per a una matriu de caràcters de mida 10x10 (és l'equivalent a l'exemple anterior).

```
int main(void)
{
  int i;
  char **a; // matriu de 10x10 de char

  a = (char **) malloc(sizeof(char *) * 10);   // Pas 1
  for(i = 0; i < 10; i++)                      // Pas 2
    a[i] = (char *) malloc(sizeof(char) * 10); // Pas 2

  a[0][0] = 0;
  a[0][0] = a[0][0] + 1;

  for(i = 0; i < 10; i++) // Pas 3
    free(a[i]);           // Pas 3
  free(a);  // Pas 4

  return 0;
}
```

Aquí es mostra un esquema del que es fa a memòria en reservar memòria per a la matriu



Es demana

1. Què és el que es fa al pas 1? I al pas 2?
2. Observar com s'allibera la memòria. Es pot fer el pas 4 abans que el pas 3?

Tingueu en compte que aquí s'ha mostrat una forma de reservar memòria de forma dinàmica a la matriu. Hi ha altres formes de fer-ho i tot depèn del context i l'aplicació en què s'utilitzarà la mariu.