Sistemes Operatius 1

Sessió de problemes – 10 d'abril

Punters a C

Els punters són una característica del llenguatge C. Poder programar amb punters permet realitzar certes operacions de forma més eficient. Determinades tasques, com la memòria dinàmica, no es pot realitzar sense l'ús de punters. Aquesta sessió se centra en l'ús de punters ja que la pràctica 3 i, en especial la pràctica 4, en fa ús.

Què és un punter? Un punter, intuïtivament, és 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 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'una variable 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`.
3. Per què és més eficient modificar els valor de la variale `var` fent servir punters en comptes d'accedir-hi directament fent servir `var[i]`?

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. Quina operació aritmètica es fa en accedir a `var[i]`?
3. Modifiqueu el programa perquè els valors del vector s'inicialitzin fent servir un punter `ptr` que s'incrementi a cada iteració.

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. Aquest 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;
}
```

Què estem fent amb aquesta operació de suma de punters? On estem emmagatzemant el valor sencer? Analitzeu bé els valor que s'imprimeixen de `ptr` i `p_int`.

(continua a la pàgina següent)

En el següent exemple és mostra un codi més complex i avançat que utilitza punters sense tipus associat. Com s'organitzen les dades a la part de "Reorganitzem les dades"? Feu un dibuix per veure-ho!

```
int main () {
   int i;
   void  *ptr, *p_ptr;
   int   *p_int;
   double *p_double;

   int var_int[1000];
   double var_double[1000];

   // Inicialitzacio

   for(i = 0; i < 1000; i++) {
       var_int[i] = 2 * i + 1;
       var_double[i] = 2.5 * i + 6.7;
   }

   // Reservem memoria

   ptr = malloc(1000 * sizeof(int) + 1000 * sizeof(double));

   // Reorganitzem les dades

   p_ptr = ptr;
   for(i = 0; i < 1000; i++) {
       p_int = (int *) p_ptr;
       *p_int = var_int[i];

       p_ptr += sizeof(int);

       p_double  = (double *) p_ptr;
       *p_double = var_double[i];

       p_ptr += sizeof(double);
   }

   free(ptr);

   return 0;
}
```