ÍNDICE

V

CLASE 5

Estos son los apuntes no oficiales del curso CC3301-1 cursado en Otoño 2022 bajo las cátedras del docente Luis Mateu B. El material usado es directamente obtenido de cátedras, en adición a los materiales del curso. [1].

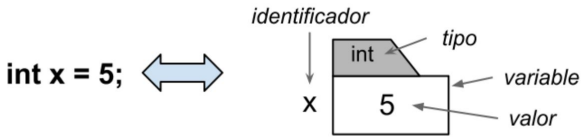
1 Agenda de la clase

- Variables
- Dirección de una variable
- Punteros
- Arreglos de variables
- Aritmética de punteros

2 Variables

Una variable almacena valores, se la conoce por su identificador (String) y su nombre es tal que le permita cambiar su valor. Otra cc es que todos los valores almacenados deben ser del mismo tipo. Gran diferencia con lenguajes dinámicamente tipados como Python (sus variables pueden almacenar valores de cualquier tipo). Tanto en C como en Java tenemos variables estáticamente tipadas.

Con las variables podemos tener operaciones. La primera de ellas es su declaración, seguida de la asignación y evaluación. Características únicas de C son las operaciones de Dirección, es decir, en donde se almacena dicha variable, porque las variables se almacenan en memoria y cada byte de la memoria tiene una dirección., así podemos saber la dirección donde comienza esta variable. Ora operación única en C es el Tamaño (sizeof(x)).



- Se conocen por su identificador
- Sirve para almacenar valores
- Todos los valores deben ser del mismo tipo
- La variable es la caja que almacena los valores
- Operaciones:

  Declaración int x;

  Asignación x=5;

  Evaluación x;

  Dirección: &x (Se suele llamar a & el operador de dirección)

  Tamaño sizeof(x)

- Regla de substitución La evaluación de una variable equivale a substituir el identificador por el valor almacenado en la variable en el momento de la evaluación. Es decir, cuando llega el momento de ejecutar

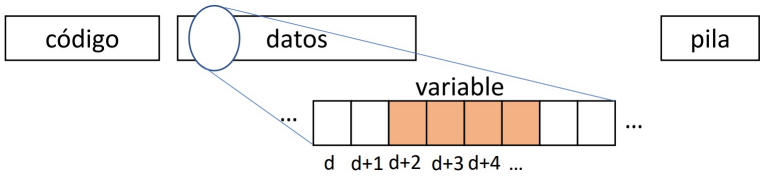
y hay una expresión que utilice la variable x, para evaluarla debemos ver el valor que tiene esta variable x (5 en el caso ejemplo) y substituir la variable por ese x.

2.1 Ejemplos

Visitemos algunos ejemplos.

```
int y = x + 10 ⇔ int y = 5 + 10
x = x + 1 ⇔ x = 5 + 1
```

3 La memoria de un programa



- El programa y sus datos se almacenan en memoria
- La memoria del programa se constituye de segmentos o áreas de memoria: código, datos, pila ...
- Cada segmento se constituye de bytes
- Cada byte tiene una dirección única
- Dentro del mismo segmento las direcciones de los bytes se asignan consecutivamente
- Una variable de tamaño X ocupa X bytes con direcciones consecutivas
- Alineamiento: Para variables de tipos primitivos (int, double, ...) la dirección de inicio es múltiplo de su tamaño

4 Punteros

- Son variables que almacenan direcciones de variables. Ejemplos:

```
int*ptr;
double *ptr_pi;
```

- Los usaremos para implementar los strings, la estructuras de datos y mucho más
- Si ptr almacena la dirección de x, se dice que ptr apunta a x
- Deben apuntar a variables del tipo declarado

```
int x;    ptr=&x;
int y;    ptr=&y;
double pi;    ptr_pi=&pi;
ptr_pi    // Incorrecto: puntero a double
          // Incorrecto: no puede apuntar a un int
```

- No es lo mismo que un entero

```
ptr = 0x7ffff8; // Incorrecto (se puede usar pero con un cast)
```

5 El operador de contenido *

- Si `x` se declaró como `int x` el tipo de `x` es `int`
- Si `s` se declaró como char * el tipo de s es char * (es para almacenar direcciones que hayan sido declaradas de tipo char)
- El símbolo * es la multiplicación cuando la operación es binaria: x * y
- Si la sintaxis es * expresión entonces es el operador de contenido y el tipo de expresión debe corresponder a un puntero como int *, double *, ...
- Usualmente la expresión es un puntero: ptr, ptr_pi
- La evaluación de *ptr en una expresión equivale a substituir *ptr por la variable a la cual apunta ptr en el momento de la evaluación

`ptr=&x;`

`*ptr=y;` ⇔ `x=y`

`ptr=&w;`

`*z=*ptr;` ⇔ `z=w`

- El tipo de *ptr es el tipo de la variable a la cual apunta ptr: si ptr es de tipo int* ⇔ *ptr es de tipo int

6 Arreglos de variables

- Una arreglo es un conjunto enumerado de variables del mismo tipo
- Se declaran con [ ]. Ejemplo:

```
int a[10]; // arreglo de 10 variables enteras
char s[20]; // 20 variables de tipo caracter
```

- En una expresión el operador binario de subindicación [ ] se usa para seleccionar una variable del arreglo
- Sintaxis: exp[exp-índice] Ejemplos: `a[7]` `s[i]`
- El tipo de exp-índice debe ser entero
- 0 es el índice de la primera variable del arreglo
- 1 es el índice de la segunda variable...
- 9 es el último índice válido para el arreglo a
- No se puede determinar el tamaño del arreglo
- No hay chequeo de índices ⇒ Segmentation fault

7 Aritmética de punteros

- double x[100];
- Para obtener la dirección de un elemento de x:

```
int i = 23;
double *p = &x[i];
```

- El identificador x representa la dirección del primer elemento:

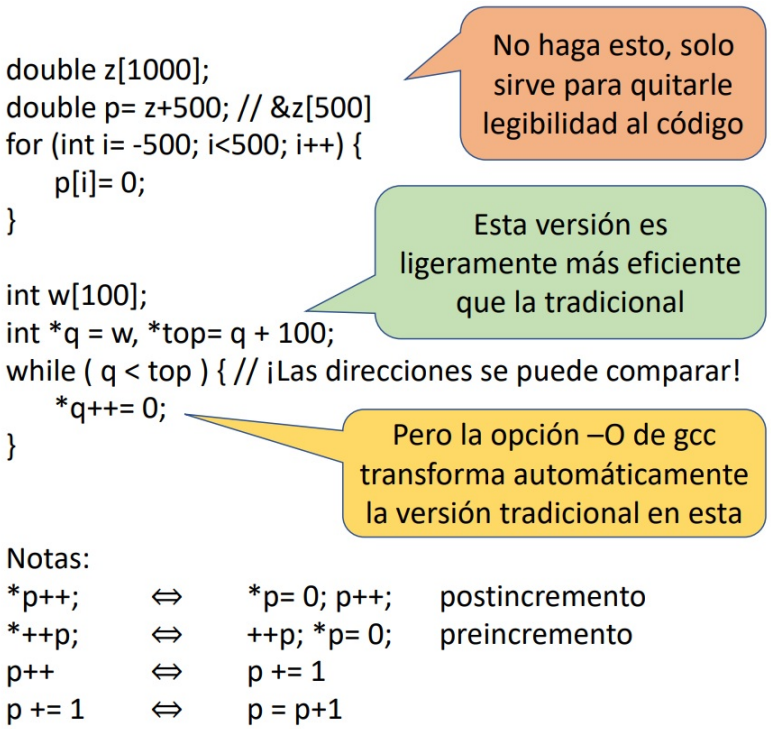
  `p = x;` ⇔ `p = &x[0];`
- En la expresión dir[índice]
  - índice puede ser cualquier expresión de tipo entero
  - dir puede ser cualquier expresión de tipo puntero (T *)
  - ¡También puede ser un puntero! Ejemplo: `p[i]`
  - Accede a la variable de dirección dir + índice * sizeof(T)
  - ¡El índice puede ser negativo!
  - Si la dirección cae fuera de un segmento del programa

    ⇒ Segmentation fault
- Equivalencias:

  & dir[índice] ⇔ dir + índice

  & dir[-índice] ⇔ dir - índice

8 Maneras rebuscadas de inicializar un arreglo en 0, pero correctas



9 Cuidado con los arreglos

La declaración: char s[10];

- Atribuye espacio en memoria para un arreglo de 10 caracteres
- s representa la dirección del primer elemento del arreglo
- Pero s no es una variable, representa un valor
- No se puede cambiar la dirección con una asignación:

```
s = ...; // ¡incorrecto!
```

- Por la misma razón que una constante no se puede cambiar con una asignación:

```
1000 = ...; // ¡incorrecto!
```

- Pero un puntero sí se puede asignar

```
char *p = s; // p almacena &s[0]
p = p + 5; // correcto porque p sí es una variable
```

- Pero la declaración solo atribuye espacio para el puntero, nunca para lo apuntado

```
char *p; p[4]= 0; // ¡incorrecto!
```

10 Más sobre arreglos y punteros

- Si p es un puntero o arreglo, las siguientes expresiones son inválidas: `p*5` `p + 2.5` `p/10`
- Solo tiene sentido p + o − una expresión entera
- Si p y q son punteros del mismo tipo: `p-q` es correcto
- Declaración con inicialización:

```
int a[ ] = { 2, 3, 5, 7, 11, 13 }; // Arreglo de 6 elementos
int b[100] = { 2, 3, 5 } ; // solo inicializa los 3 primeros
                           // de un arreglo de 100 elementos
b = { 4, 8, 12 }; // Incorrecto, no es una declaración
int c[ ]; // Incorrecto, no se puede inferir el tamaño
```

- Una función no puede recibir como parámetro un arreglo, pero sí puede recibir un puntero y operarlo como si fuese un arreglo
- Si se declara una función con este encabezado: `void fun(int arreglo[ ]);`
-
- El compilador silenciosamente lo cambia por: `void fun(int *arreglo);`

11 Resumen

- Una variable reside en la memoria
  - Almacena valores de un tipo específico
  - Posee un tamaño en bytes y una dirección
- Un puntero es una variable que almacena direcciones de variables
  - Sirven para implementar los strings y las estructuras de datos
  - El operador de contenido permite acceder a la variable a la cual apunta un puntero
- Un arreglo es un conjunto enumerado de variables del mismo tipo
  - El operador de subindicación permite seleccionar una de esas variables por medio de un índice
  - Punteros y arreglos son similares, pero hay diferencias
  - Hay una equivalencia entre subindicación y aritmética de punteros
- Los programas son más eficientes en C porque no hay chequeo de índices, pero los errores de manejo de C son difíciles de depurar

12 Código usado en clases

```c
#include <stdio.h>

int main() {
  int x;                // declaracion
  x = 5;                // asignacion
  printf("%d\n", x);    // lectura
  printf("%p\n", &x);   // direccion
  printf("%ld\n", sizeof(x)); // Tamano
  int y = x + 10;       // declaracion con
                        // inicializacion
  x = x+1;              // asignacion
  double pi = 3.14159; // variable real
  pi = pi + x;
  printf("%ld\n", sizeof(pi));

  int *ptr = & x;        // decl. puntero
  printf("%p\n", ptr);
  ptr = &y;
  double *ptr_pi = &pi;
  printf("%p\n", ptr_pi);

  *ptr = x;      // operador de contenido
                 // en una asignacion
  x = 1;
  int z = *ptr; // operador de contenido
                 // en una expresion
  printf("%d\n", z);

  int a[10];
  printf("%p\n", a);
  for (int i= 0; i<10; i++) {
    a[i]= i*i;
    printf("%p: %d\n", &a[i], a[i]);
  }
  int k= 1000000000;
  printf("%d\n", a[k]);

  return 0;
}
```

REFERENCIAS

[1] Mateu, Luis: *Programación de Software de Sistemas - Novedades.* `https://www.u-cursos.cl/ingenieria/2022/1/CC3301/1/novedades/`, 2022.