Contenido

Introducción al lenguaje C*

El objetivo de este documento es mostrar las principales características del lenguaje que será utilizado en el curso y ver sus principales diferencias con Pascal.

El lenguaje C* no es un lenguaje real, es la denominación que utilizamos en el curso para referirnos al lenguaje C sumándole algunas pocas cosas de C++.

hola.pas:

```
program Hola;
begin
    writeln('¡Hola, mundo!');

end.
```

hola.cpp:

```
#include <stdio.h>

int main()
{
    printf("¡Hola, mundo!\n");
    return 0;
}
```

Todo programa en C/C++ necesita implementar la función *main*. *Main* es una función especial, a partir de la cual comienza la ejecución del programa.

Compilación

Para poder utilizar las características de C++ los archivos tienen extensión .cpp.

Los programas en C/C++ se componen por varios archivos, ya sean los compilados por el desarrollador como las bibliotecas pre compiladas. Las bibliotecas son módulos que incluyen funciones y constantes que el usuario puede utilizar en sus programas, para esto debe incluir la biblioteca.

C/C++ permite compilar los archivos de forma separada. Para compilar ejecutamos en la consola el comando:

```
g++ -c holamundo.cpp
g++ -c programa.cpp
```

esto genera un nuevo archivo con extensión.o (holamundo.o). El .o no es un ejecutable. Para obtener el ejecutable necesitamos linkear todos los archivos que forman el programa y las bibliotecas utilizadas. Para linkear los archivos en consola debemos ejecutar:

```
g++ holamundo.o programa.o -o nombre_ejecutable
```

Esto genera el ejecutable con nombre “nombre_ejejcutable”.

Nota: las bibliotecas se asocian de forma automática a partir de las referencias en el código.

La compilación y linkeo puede hacerse con un solo comando:

```
g++ holamundo.cpp programa.cpp -o nombre_ejecutable
```

Bibliotecas

Para utilizar una biblioteca se la incluye al inicio del archivo utililzando *#inlcude*.

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

Este ejemplo incluye la biblioteca *stdio*, que es la que proporciona las operaciones para interactuar con la entrada y la salida estándar (veremos más adelante).

Tipos de datos

Pascal	C*
INTEGER	Int (C/C++)
REAL	float (C/C++)
CHAR	char (C/C++)
BOOLEAN	bool (C++)

Variables

Las variables pueden ser declaradas en cualquier lugar del programa. Es recomendable declararlas cuando se usan por primera vez. Los nombres de las variables deben ser alfanuméricos y pueden incluir el carácter '_'. Deben comenzar con una letra o con '_'.

C* es case sensitive, es decir que la variable *nombre* es diferente que la variable *Nombre*.

```
int numero;
float real;
char c;
bool bandera;
```

En C* el operador de **asignación** es '='. Cuando declaramos una variable debemos inicializarla para que su valor inicial no quede sujeto a las diferentes versiones del compilador.

```
numero = 3; //asigno el valor de 3 a numero
c = 'A'     //asigno el caracter A a c
int b = 7;  //declaro b y le asigno 7
```

La asignación retorna un valor, por lo que es válido: a = b = 9.

Operadores

- Comparación

C*	Acción
==	Igual
!=	Distinto
>	Mayor
<	Menor
>=	Mayor Igual
<=	Menor Igual

- Lógicos

C*	Tipo	Acción
&&	Binario	Conjunción (Y)
||	Binario	Disyunción (O)
!	Unario	Negación

En C* los operadores binarios se evalúan de forma perezosa (lazy evaluation) o por circuito corto. Significa que siempre se intenta hacer el mínimo esfuerzo posible.

En la siguiente expresión si *b* no es mayor que 3, la comparación *c=='A'* no se hace y toda la expresión evalúa *false*.

```
bandera = (b > 3) && (c == 'A');
```

En el siguiente ejemplo, si b es menor que 7 la comparación c=='D' no se hace y toda la expresión evalúa a *true*.

```
bandera = (b < 7) || (c == 'D');
```

- Aritméticos

C*	Acción
+	Suma
-	Resta
/	División entera o real (depende del tipo de los operandos)
%	Modulo (resto de la división entera)

Otros operadores útiles:

- Incremento en uno (++):
    - tanto a++ como ++a incrementan el valor de a en una unidad
    - sin embargo, cuando asignamos:
        - x=a++; incrementa el valor de a y retorna su valor antes del incremento. Equivale a las sentencias: x=a; a=a+1;
        - x=++a; incrementa el valor de a y retorna el valor incrementado. Equivale a las sentencias a=a+1; x=a;
- Decremento en uno (--):
    - Análogo al incremento.
- Incremento en x: a+= x incrementa el valor de a en x (equivalente a=a+x)
- Decremento en x: a-= x decrementa el valor de a en x (equivalente a=a-x)
- Multiplicar por x: a*= x multiplica a por x y lo almacena en a (equivalente a=a*x)
- División por x: a/=x divide a entre x y lo alacena en a (equivalente a a=a/x)

Comentarios

Se pueden hacer comentarios de bloques (C/C++) o de líneas (C++).

```
/* esto es un bloque
comentado, observe los
operadores de apertura y cierre*/

int x; //esto es una linea comentada
```

Constantes

Existen dos posibilidades de definir constantes, definirlas al inicio del programa utilizando *define (*C) o definirlas en cualquier momento utilizando *const* (C++).

Con define:

```c
#define BASE 10
#define ALTURA 5
int main() {
    int area = BASE * ALTURA;
    printf("Area: %d", area);
    return 0;
}
```

Con const:

```c
int main() {
    const int BASE = 10;
    const int ALTURA = 5;
    int area = BASE * ALTURA;
    printf("Area: %d", area);
    return 0;
}
```

La diferencia es que *define* es una directiva al precompilador que produce un reemplazo de texto antes de compilar y *const* utiliza variables (y por lo tanto tiene su espacio de memoria, su tipo, etc.) que no se pueden modificar.

Nota: Es buena práctica definir los nombres de las constantes en mayúsculas y el de las variables en minúsculas.

Estructuras de control

Selección

El lenguaje C nos proporciona 2 estructuras de selección: **if-else** (puede no llevar el else) y **swicth**. Su funcionamiento es similar al de las estructuras de Pascal, donde switch equivale a case.

if-else:

```
if (6 <= valor && valor <= 12) {
    printf("Aprobado"); cantidad_aprobados++;
} else if (valor >= 3) {
    printf("Examen");
} else if (valor >= 0){
    printf("Reprobado");
} else{
    printf("Valor incorrecto");
}
```

switch:

```
switch (valor){
    case 6: case 7: case 8: case 9: case 10: case 11: case 12:
        printf("Aprobado");
        cantidad_aprobados++;
        break;
    case 3: case 4: case 5:
        printf("Examen");
        break;
    case 0: case 1: case 2:
        printf("Reprobado");
        break;
    default: printf("Valor incorrecto");
}
```

Nota: Si se omiten los **breaks** la ejecución continúa evaluando los case posteriores al que ingresa primero. Por ejemplo, si *valor* es 3 ejecutaría el case de 3 y evaluaría la condición el case de cero.

Iteración

Para la iteración en C contamos con **for**, **while** y **do-while**, este último es equivalente a REPEAT.

- **While**

```
while (condición)
    cuerpo
```

```c
int i = 0;
while (i < 10) {
    printf("*");
    i++;
}
```

- **For**

```
for (inicio; condición; paso)
    cuerpo
```

En C/C++ la instrucción *for* admite como condición cualquier expresión booleana. Por ejemplo expresiones compuestas con partes que no dependen del contador.

```c
//con contador
for (int i = 0; i < 10; i++){
    printf("*");
}

//con expresión booleana
bool esPar = false;
for (int i = 0; i < 10 && !esPar; i++){
    int a;
    scanf("%d", &a);
    esPar = (a%2) == 0;
}
```

- **Do-while**

  do

  cuerpo

  while (condición)

```
int i = 0;
do{
    printf("*");
    i++;
}while (i < 10);
```

Enumerados

Los enumerados se definen utilizando *enum*.

```
enum mes {enero, febrero, marzo, abril, mayo, junio, julio, agosto, setiembre, octubre, noviembre, diciembre};

mes este_mes = marzo;
```

Los valores de los enumerados se mapean con los números enteros comenzado por el 0, es decir que enero = 0. Por ejemplo, si se imprime la variable *este_mes* en pantalla se muestra 2 (marzo).

Estructuras

Las estructuras son similares a los registros *record* de Pascal. Se definen utilizando *struct*.

```
struct fecha {
    int f_dia;
    mes f_mes;
    int f_anio;
};
```

Para acceder a los miembros se utiliza el punto.

```
fecha f;
f.f_dia = 22;
f.f_mes = enero;
f.f_anio = 2019;
```

Nota: esta es la forma de declarar estructuras y enumerados en C++ (y en C*).

Punteros

El manejo de memoria en C* lo haremos como se hace en C++, mediante el uso de *new* y *delete* para reservar y liberar memoria respectivamente.

El operador de referencia (*) permite acceder a la información almacenada en la memoria referenciada por un puntero. Mientras que el operador de deferencia (&) nos permiten obtener la dirección de memoria de una variable.

Cuando tenemos un puntero existen 4 formas de inicializarlo:

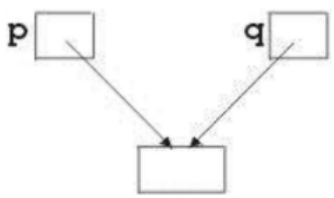
- Asignarle NULL (NULL es una constante que vale 0).
- Asignarle un nuevo lugar de memoria mediante el uso de *new*. Que no contendrá un valor definido por el usuario.
- Asignarle la dirección de memoria de una variable existente utilizando el operador *&*.
- Asignarle la dirección de memoria referenciada por otro puntero (alias).

```
int * p; // p es un puntero a un número entero
int i; p = &i; // p apunta a la dirección de i
*p = 10; // i toma el valor 10
int* p2; p2 = p; // p2 apunta a la dirección de i
p = NULL; // así se deja en NULL un puntero
p = new int; // así se pide memoria
delete p; //libera la memoria apuntada por p
```

Nota: estaría mal liberar la memoria asignada a p2 ya que no es memoria asignada dinámicamente.

Cuando se tienen dos punteros que apuntan a la misma memoria, se les llama alias. A la hora de liberar la memoria se debe tener esto presente.

Dada la situación:



¿Qué ocurre con q si hacemos `delete p;`?

q queda apuntando a un espacio de memoria que ya no pertenece al programa.

También se debe considerar que cuando tenemos alias no se pueden eliminar todos. En el ejemplo, hacer `delete q` retornaría un error ya que q no hace referencia a memoria perteneciente al programa.

Cuando tenemos un puntero a una estructura podemos acceder a sus campos de las siguientes formas:

```
(*puntero_fecha).f_dia = 4;
puntero_fecha->f_dia = 4; // más fácil
```

Arreglos

Son una colección de elementos del mismo tipo de datos almacenados uno a continuación del otro. Para acceder a las diferentes posiciones del arreglo contamos con índices. En C/C++ los índices siempre se numeran desde 0. Un arreglo de tamaño N tendrá como último índice N-1. Indicar un índice fuera del rango puede causar un error de acceso inválido de memoria o "Segmentation fault".

Tenemos dos formas de declarar arreglos:

- de forma estática (durante la compilación del programa mediante gestión estática de memoria: declaración de variables)
- de forma dinámica (durante la ejecución del programa mediante gestión dinámica de memoria: punteros).

Estaticos:

```
int arr[2]; // las celdas estan indefinidas. Posibles arr[0] y arr[1]
int vector[5] = {1, 2, 3, 4, 5}; //inicializado con esos valores

//asigna 7 en la posicion 2
vector[2] = 7; //el arreglo queda {1, 2, 7, 4, 5}

int matriz[2][3] = {{1, 2, 3}, {4, 5, 6}};
//asigna 10 en la fila 2, columna 1
matriz[2][1] == 10; // queda {{1, 2, 3}, {10, 5, 6}}
```

Dinámicos:

```
int* vector = new int[10]; //equivalente a vector[10]
delete [] vector;
```

El tamaño al momento de la definición debe ser un valor que puede indicarse por medio de una variable, constante o una expresión aritmética.

```
int a[N];
int b[c+3];
```

Conversión de tipos

La mayoría de las conversiones de tipos se dan de forma implícita, es decir que C/C++ lo resuelve.

```
float vf = 1.6;
int vi = 1 + vf; // vi = 2 (float se trunca)
vi = 1 + vf + vf; // vi = 4 (cast al "más grande")
vi = vi + true; // vi = 5 (true es 1)
vi = vi + false; // vi = 5 (false es 0)
vi = 'a' + 1; // vi = 98 (valor ASCII)
char vc = 'a' + 1; // vc = 'b'
vf = 1.5 + vi // vf = 99.500000
bool vb = 237; // vb = true (0 es false, otro true)
vf = 3 / 2; // vf = 1.000000
```

Pero también podemos hacerla de forma explícita, lo que se denomina casteo (casting).

```
//con casting
vf = (float)3 / 2; // vf = 1.500000

//sin casting
vf = 3 / 2; // vf = 1.000000

//casting de toda la expresion
vf = (float)(3 / 2); // vf = 1.000000
```

Observar que en el ejemplo sin casting la división que se hace es entera, ya que ambos operadores son enteros por esto el resultado es 1. Mientras que en el ejemplo con casting la división es real, ya que indicamos que el 3 es un real.

Funciones y procedimientos

Las funciones en C/C++ se definen indicando su tipo de retorno, nombre y entre paréntesis los parámetros separados por coma.

```
int suma (int a, int b){
    int res = a+b;
    return res;
}
```

Return finaliza la ejecución de la función y devuelve el valor al lugar donde fue invocada la función. Para invocar una función se puede hacer así:

```
sum = suma(3, 5);
```

Nota: Cualquier sentencia indicada después de *return* no se ejecutará.

En C/C++ no existe una forma explicita de definir un procedimiento. Le llamamos procedimiento a las funciones que no retornan un valor. Para indicar esto usamos el tipo *void*.

```
void imprimirSuma(int a, int b) {
    int suma = a + b;
    printf("La suma es: %d\n", suma);
}
```

Las funciones deben declararse fuera de cualquier otra función (las declaraciones no se pueden anidar).

Pasaje de parámetros

En C todos los parámetros se pasan por valor. C++ permite el pasaje de parámetros por referencia mediante el uso del operador &. En C* utilizaremos el pasaje por referencia de C++.

```
void suma (int a, int &b){
    b = a+b;
}
```

Observar que no se utiliza *return*.

En C el pasaje por referencia se simula utilizando punteros.

```
void inc(int *i) {
    (*i) += 1;
}

//se debe invocar de esta manera
int x = 2;
inc (&x);
```

Entrada y Salida

La entrada y salida estándar en C se hace con las funciones *printf* y *scanf* respectivamente. Ambas son funciones especiales que reciben una cantidad variables de parámetros. El primer parámetro es la **cadena de texto de formato**, el resto depende de los **especificadores de formato** que se encuentren en el primero.

Especificadores:

Estos son algunos de los indicadores más utilizados.

- %d int
- %c char
- %f float
- %s char*

```
printf("hola mundo\n");
printf("-> %f, ", estructura->dato); //estructura->dato es un float
int pri = 1;
int seg = 2;
printf("(%d, %d)\n", pri, seg); //imrpime (1,2) y un enter
```

scanf utiliza la cadena de formato al igual que *printf*, pero los parámetros deben ser punteros a las variables donde quedarán almacenados los valores leídos de la entrada.

```
int val, cant;
char str [10];
cant = scanf("%d-%s", &val, str);
```

Nota: para pasar un puntero a una variable estática se utiliza &.

Ambas funciones están en la biblioteca estándar de C, en stdio.h (“Standard Input-Output” o Entrada y salida estándar) Para poder usarlas, se debe importar la biblioteca:

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

En C++ existen otras alternativas (p.ej. cin y cout) para manejar la entrada y salida estándar, pero en este curso recomendamos usar el estilo C.

Resumen

En resumen, C* es C pero con las siguientes cosas agregadas de C++:

- Uso de new y delete
- Comentarios por línea
- Declaración de tipos como en C++ para registros y enumerados
- Pasaje por referencia
- El tipo de datos *bool*.