SISTEMAS OPERATIVOS

TALLER No. 1: Toma de contacto con el lenguaje C

OBJETIVO

Familiarizar al estudiante con el lenguaje de programación C: estructura básica de un programa, algunos elementos básicos del lenguaje, sintaxis, escritura de código fuente y compilación.

ACLARACIÓN

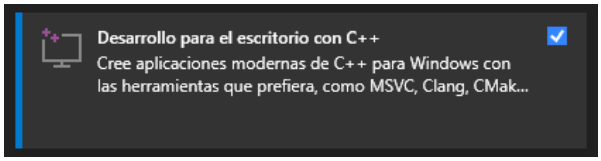
Este taller **NO** es un tutorial sobre el lenguaje C. En este taller se exponen algunos elementos claves que el(la) estudiante debe tener presentes cuando escribe programas en C. Se invita a al(la) estudiante a que explore y estudie por su propia cuenta aspectos que no se abordan en este primer taller: ciclos, arreglos, estructuras de control, tipos de datos extendidos, entre otros. El(La) estudiante que ya conoce lenguajes de programación como Java, C++ o incluso C#, encontrará muchas similitudes con el lenguaje C.

CONTENIDO DEL TALLER

1. Descarga e instalación de Visual Studio Community.
2. Verificar funcionamiento del compilador de C/C++.
3. Escribir un primer programa en C.
4. Compilación del primer programa.
5. Responder preguntas con relación al proceso de compilación.
6. Modificar el proceso de compilación.
7. Responder preguntas con relación a los dos procesos de compilación.
8. Ejecutar el enlazador de C/C++.
9. Responder preguntas con relación a los procesos de compilación y enlace.
10. Tipos de datos, declaración de variables y salida de datos por pantalla.
11. Trivia.
12. Entrada de datos.
13. Inténtelo usted.
14. Operadores booleanos.
15. Conceptos básicos sobre apuntadores en el lenguaje C.
16. Inténtelo usted.
17. Prueba de escritorio.
18. Ejemplo de paso de parámetros a funciones usando apuntadores.
19. Compiladores de C/C++ adicionales para Windows y Linux.

DESARROLLO DEL TALLER

1. Descargar e instalar Visual Studio Community 2019 o 2022 desde el siguiente URL y asegúrese de seleccionar las opciones de desarrollo con C++ (Desarrollo para el escritorio con C++).

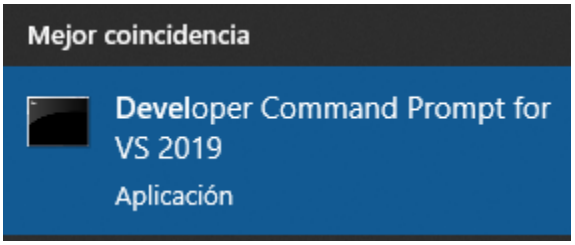


   URL de descarga: https://visualstudio.microsoft.com/es/vs/community/

   NOTA: En este primer ejercicio de toma de contacto con el lenguaje C, nos interesa el compilador de C/C++, más que el entorno de desarrollo de aplicaciones de Visual Studio Community.

2. Verificar que tiene el compilador de C/C++ instalado a través de Microsoft Visual Studio Community. Para verificar que tiene instalado el compilador de C/C++ de Microsoft, siga los siguientes pasos.

   2.1 Sobre la casilla de búsqueda del menú inicio, busque con la siguiente expresión "**Developer Command Prompt**". La siguiente imagen proporciona un ejemplo.



   2.2 Haga clic para abrir la línea de comandos para desarrolladores de Visual Studio.

   2.3 En la línea de comandos para desarrolladores de Visual Studio, ejecute el comando `cl` y si todo está correctamente instalado, debe obtener un resultado similar al siguiente.

```
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community>cl
Compilador de optimización de C/C++ de Microsoft (R) versión 19.29.30140 para x86
(C) Microsoft Corporation. Todos los derechos reservados.

uso : cl [ opción... ] nombre de archivo... [ /link opción de vínculo... ]
```

3. Abra un editor de texto y copie el siguiente código. Guarde el archivo con el nombre **holamundo.c**. Atento(a) a la extensión del archivo: por convención, los archivos de código fuente del lenguaje C, llevan extensión **.c**.

   IMPORTANTE: Tenga presenta la ubicación en donde está guardando el archivo de texto que acaba de crear.

```c
// Incluir la libreria estandar para manejo de I/O
#include <stdio.h>

// Programa principal
int main(void) {

    // printf() para generar la salida por pantalla
    printf("Hola mundo desde C!");
    return 0;
}
```

4. Compile el programa anterior con el compilador **cl** que verificó en el numeral 2 de este documento.

   4.1 Abra la línea de comandos para Desarrolladores de Visual Studio.

   4.2 Ubíquese en el directorio (carpeta) en donde guardó el archivo **holamundo.c**. Utilice el comando **cd** del sistema operativo para cambiarse al directorio en donde guardó el archivo **holamundo.c**. Por ejemplo, si el archivo se guardó en el escritorio del usuario **pepe**, tendría que ejecutar lo siguiente.

```
> cd c:\Users\pepe\Desktop
```

   IMPORTANTE: El símbolo **>** representa el *prompt* de la línea de comandos. **NO** copie este símbolo cuando reproduzca estas acciones.

   4.3 Ejecute la orden de compilación del código fuente pasando a la herramienta **cl**, el nombre del archivo **holamundo.c**.

```
> cl holamundo.c
```

   4.4 Ejecute el programa que acaba de crear escribiendo lo siguiente en la línea de comandos.

```
> holamundo.exe
```

5. Responda las siguientes preguntas.

   5.1 ¿Qué genera **cl** al compilar su programa?
   5.2 ¿Cuál es el nombre del archivo (y extensión) que incluye el código fuente de C?
   5.3 ¿Cuáles son los nombres de los archivos que produce **cl**?
   5.4 ¿Cuál es la ubicación de los archivos relacionados con este primer ejercicio?
   5.5 ¿Cuál es el nombre del archivo ejecutable?

6. Elimine los archivos **producidos** por la herramienta **cl**. Ejecute de nuevo la herramienta **cl** de la siguiente manera.

```
> cl /c holamundo.c
```

7. Responda las siguientes preguntas

   7.1 ¿Qué nota de diferente con relación al resultado obtenido en el numeral 4?
   7.2 ¿Qué archivos se producen con esta opción?
   7.3 ¿Tiene manera de ejecutar su programa? ¿Por qué?

8. Ejecute el enlazador de C/C++ de la siguiente manera.

```
> link holamundo.obj
```

9. Responda las siguientes preguntas.
   9.1 ¿Qué archivo(s) se producen como resultado de la ejecución anterior?
   9.2 ¿Tiene manera de ejecutar su programa?
   9.3 ¿Qué podría concluir de todo lo ocurrido hasta este punto usando **cl**, **cl /c** y **link**?

10. Tipos de datos en, declaración de variables y salida de datos por pantalla: para esta primera toma de contacto con el lenguaje C, algunos tipos de datos, declaraciones de variables y salida de datos por pantalla se incluyen en el siguiente programa. **Escriba, compile y ejecute el siguiente programa.**

```c
#include <stdio.h>

int main(void) {

    int mientero = 5;
    float mifloat = 5.0;
    double midouble = 3.141516;
    long double milongd = 3.1414E10;
    char michar = 'c';
    char otrochar = 99;
    signed short int misint = -10;
    char mistring[] = "Hola";

    printf("Hola mundo desde C\n");
    printf("mientero = %d\n", mientero);
    printf("mifloat = %f\n", mifloat);
    printf("midouble = %lf\n", midouble);
    printf("milongdouble = %lf\n", milongd);
    printf("michar = %d\n", michar);
    printf("michar = %c\n", michar);
    printf("otrochar = %c\n", otrochar);
    printf("misint = %d\n", misint);
    printf("mistring = %s\n", mistring);
    return  0;
}
```

La declaración de variables tiene las siguientes variaciones en su sintaxis.

Sintaxis	**Ejemplo**
<tipo de dato> <variable>;	int edad;
<tipo de dato> <variable> = <valor inicial>;	char c = ‘y’;
<tipo de dato> <variable 1>, <variable 2>, <variable 3>;	short a, b, c;

La función **printf()** se usa para escribir los datos contenidos en las variables. Por defecto, se usa la salida estándar para escribir los datos de las variables a la pantalla. Para poder usar esta función se deben usar especificadores de formato para los datos que se desea escribir. Algunos especificadores de formato se resumen en la siguiente tabla.

Tipo de dato	Especificador de formato
int	%d
char	%c
float	%f
double	%lf
short int	%lf
unsigned int	%u
long int	%li
long long int	%lli
unsigned long int	%lu
unsigned long long int	%llu
signed char	%c
unsigned char	%c
long double	%Lf

Teniendo presente los modificadores en la función **printf()**, preste atención al resultado de las líneas 19, 20 y 21**. ¿Qué puede concluir del resultado en estos tres casos?**

11. Trivia: resultado de impresión y tipo de dato **size_t**.
    - 11.1 ¿Qué imprime el siguiente programa?
    - 11.2 ¿Qué es el tipo de dato **size_t**?
    - 11.3 ¿Qué hace la función **sizeof()**?

```
1  #include <stdio.h>
2
3  int main(void) {
4
5      signed short int ssint = -10;
6      long longint = 10;
7      size_t tssint = sizeof(ssint);
8      size_t tlint = sizeof(longint);
9      printf("tssint = %zd\n", tssint);
10     printf("tlint = %zd\n", tlint);
11     return  0;
12 }
```

12. Entrada de datos. La entrada de datos en C se comúnmente se realiza con la función `scanf()`. De manera similar a la función `printf()`, se deben usar especificadores de formato para especificar que el dato leído se debe almacenar en una variable de cierto tipo de dato. **Escriba, compile y ejecute los siguientes programas.**

```c
#include <stdio.h>

int main(void) {
    int edad;
    printf("Ingrese su edad: ");
    scanf("%d", &edad);
    printf("edad = %d\n", edad);
}
```

```c
#include <stdio.h>

int main(){
    int a;
    float b;
    printf("Ingrese un entero y un float: ");
    scanf("%d%f", &a, &b);
    printf("a = %d, b = %f", a, b);
    return 0;
}
```

En algunos casos, a la función `scanf()` **hay que pasarle la dirección de memoria de la(s) variable(s)** en donde será guardado el dato leído. Esto se hace con el operador `&` y esta es la razón del uso de este símbolo junto a las variables que se pasan a la función `scanf()`.

13. Inténtelo usted: escriba un programa en C que lea su nombre y lo muestre por pantalla.

14. Operadores booleanos. En el lenguaje C se tienen tres tipos de operadores booleanos.

Operador	Descripción
&&	Conjunción lógica (Y/AND)
||	Disyunción lógica (O/OR)
!	Negación (NO/NOT)

Escriba, compile y ejecute el siguiente programa.

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

int main(void){
    bool t = true;
    bool f = false;
    printf("!true: %d\n", !t);
    printf("true && false: %d\n", t && f);
    printf("true && !false: %d\n", t && !f);
    printf("true || false: %d\n", t || f);
    printf("false || false: %d\n", f || f);
    return 0;
}
```

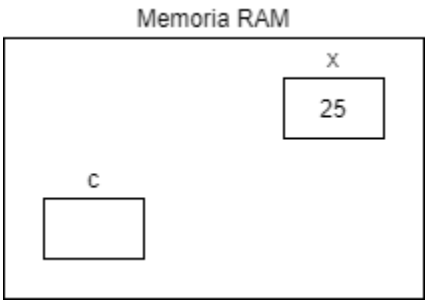
15. Conceptos básicos sobre apuntadores en el lenguaje C.

A diferencia de otros lenguajes de programación, en el lenguaje C, los apuntadores son explícitos y saber usarlos es fundamental. El apuntador **es una variable** que, en lugar de almacenar un dato como la edad, un valor real o un valor entero por poner solo algunos ejemplos; el apuntador **almacena una dirección de memoria de una variable**.

Suponga que tenemos declaradas las siguientes variables.

```c
int x = 25;
char c;
```

La siguiente figura representa una analogía de la memoria RAM de un computador, las variables **x** y **c** quedaron en posiciones aleatorias de la memoria RAM. Las posiciones en donde están las variables **x** y **c**, se identifican por su dirección de memoria.



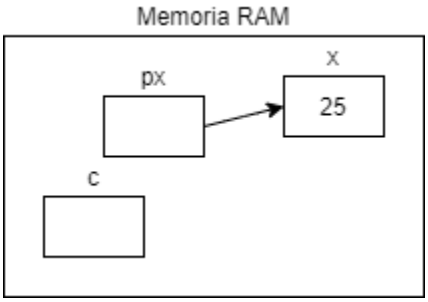
Declaración de un apuntador: una variable que almacena una dirección de memoria de otra variable (un apuntador), siempre **se debe declarar del mismo tipo de dato que la variable a la cual apunta**.

```
<tipo de dato de la variable al cual apunta> * <nombre del apuntador>;
```

Ejemplo:

```
int *px = &x;
```

Se está declarando la variable `*px` que **es un apuntador a la variable** `x`. Para almacenar la dirección de memoria de `x` en `*px` es necesario indicarlo con el operador `&`: `&x` está indicando la dirección de memoria de la variable `x`. La siguiente figura ilustra gráficamente la analogía con el estado de la memoria RAM luego de la instrucción anterior.



Analice y concluya sobre el siguiente programa.

```c
#include <stdio.h>

int main(void){
    int x = 25;
    char c;

    int *px = &x;
    char *pc = &c;

    printf("x = %d; px = %p\n", x, px);
    printf("x = %d; *px = %d\n", x, *px);
    printf("&x = %p; px = %p\n", &x, px);
}
```

Se debe tener claro lo siguiente.

- `int *px` es la declaración de una variable (`px`) apuntador.
- `*px = &x` es la asignación de la dirección de memoria de `x` a `px`.
- `px` almacena la dirección de memoria de la variable `x`.
- `*px` hace referencia al contenido de la dirección almacenada en `px`.
- `&x` es la dirección de memoria de la variable `x`.

En general: `px` es una dirección de memoria y `*px` es del dato almacenado en dicha dirección de memoria.

16. Inténtelo usted: usando el programa anterior, súmele 25 a la variable `x` usando el apuntador definido. No lo haga directamente sobre la variable `x`. Repita los tres `printf()` luego de esta suma y comprenda el resultado de todas las operaciones.

17. **Sin usar el compilador**, explique el efecto de cada una de las siguientes instrucciones (prueba de escritorio).

```
1   #include <stdio.h>
2
3   int main(void){
4       int x, *px;
5
6       px = &x;
7       *px = 0;
8       *px = *px + 3;
9       *px = *px * 2;
10      x = *px + 1;
11      *px *= x * 2;
12      printf("%d %d\n",x, *px);
13      printf("%d %p\n",x, px);
14  }
```

Línea 4	
Línea 6	
Línea 7	
Línea 8	
Línea 9	

Línea 10	
Línea 11	
Línea 12	
Línea 13	
Línea 14	

18. Analice el siguiente programa, comprenda lo que se está haciendo: ¿qué tipo de paso de parámetros se está haciendo en este caso a la función `cambio()`?

```c
#include <stdio.h>

void cambio(int *, int *);

int main(void){
    int x, y;
    x = 3;
    y = 2;
    printf("x = %d, y = %d\n",x, y);
    cambio(&x, &y);
    printf("x = %d, y = %d\n",x, y);
}

void cambio(int *a, int *b) {
    *a += *b;
    *b = *a - *b;
    *a -= *b;
}
```

¿Qué cambios haría usted en la función `cambio()` para que los valores originales pasados a la función no se pierdan?

19. Indague qué otros compiladores (gratuitos/open source) podría instalar y usar en Windows y Linux.