Licenciatura en Sistemas
Programación de computadoras

Equipo docente: Jorge Golfieri, Natalia Romero, Romina Masilla y Nicolás Perez
Mails: jgolfieri@hotmail.com , nataliab_romero@yahoo.com.ar , romina.e.mansilla@gmail.com, nperez_dcao_smn@outlook.com
Facebook: https://www.facebook.com/groups/171510736842353
Git: ***http://***github.com/UNLASistemasProgramacion/Programacion-de-Computadoras

---

Unidad 1:

Resolución de problemas. Fase de resolución del problema: Análisis del Problema, Diseño del Algoritmo. Verificación de algoritmos. Fase de Implementación. Prueba de escritorio. Conceptos de compiladores e intérpretes. Lenguajes de programación. Lenguaje C. Tipo de datos en C. Constantes y Variables. Operadores y expresiones. Problemas: implementación y testeo.

Bibliografía citada:

Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos (2a. ed.) Luis Joyanes Aguilar, Luis Rodríguez Baena y Matilde Fernández Azuela. McGraw-Hill España. Capítulo 1-2-3.

C algoritmos, programación y estructuras de datos. Luis Joyanes Aguilar, Andrés Castillo Sanz, y Lucas Sánchez García. Capítulo 1-2-3-4.

Unidad 1: Resolución de problema en base a la programación de computadoras

¿Qué es la programación?:

Si bien ya lo vimos en el curso de ingreso, no está de más recordar que, no es otra cosa más que ordenar las acciones a cumplir para lograr un determinado objetivo. El orden en que se realizan las acciones y las acciones propiamente dichas son los que se conocen como el algoritmo, es decir el recetario de instrucciones.

Un algoritmo es una secuencia no ambigua, finita y ordenada de instrucciones que han de seguirse para resolver un problema. Un programa normalmente implementa (traduce a un lenguaje de programación concreto) uno o más algoritmos. Un algoritmo puede expresarse de distintas maneras: en forma gráfica, como un diagrama de flujo, en forma de código como en pseudocódigo o un lenguaje de programación, en forma explicativa.

Nuestra materia se separa en dos partes, una parte relativamente teórica los lunes y los viernes en el pizarrón, donde pensaremos utilizando pseudocódigo y más adelante C; y una parte práctica, los martes y/o miércoles donde se trabajará frente a la PC programando en C, utilizando el programa CodeBlocks.

¿Cómo es el pseudocódigo?:

El pseudocódigo es una transición entre el lenguaje natural (Castellano en nuestro caso) y el lenguaje que sabe manejar una PC, en nuestro caso C.

Para entender el pseudocódigo es bueno realizar un ejemplo muy sencillo programando algún algoritmo que estemos acostumbrados a realizar todos los días, como tomar un té, cepillarnos los dientes, aprobar una materia, o pasar un fin de semana haciendo algo.

Ejemplo: Realizar la suma de dos enteros en pseudocódigo.

ALGORITMO: Sumar;
PRE CONDICIONES: Saber sumar
POST CONDICIONES: Se quiere la suma de cualquiera de dos números enteros

VAR

ENTERO Numero1, Numero2, Resultado ← 0;

ALGORITMO

ESCRIBIR ("Dime el número1 para sumar: ");
LEER(Numero1);
ESCRIBIR ("Dime el número2 para sumar: ");
LEER(Numero2);

Resultado ← Numero1 + Numero2;
ESCRIBIR ("La suma es: ", Resultado);

FIN_ALGORITMO

Ejemplo: Realizar el pseudocódigo de un programa que permita calcular el área de un rectángulo. Se debe introducir la base y la altura para poder realizar el cálculo.

Programa; área del triangulo
Pre Condiciones: Conocer la formula del área del triangulo
Post Condiciones: Quiero el área del rectángulo

Var:
Decimal base ←0
Decimal altura ← 0
Decimal área ← 0

Algoritmo:

escribir “Introduzca la base y la altura”

leer base, altura

calcular area = base* altura

escribir “El área del rectángulo es “ area

Fin_programa

Tabla de pseudocódigo para la materia:

Pseudocódigo	**Descripción**
comentario	/* el comentario abre con barra, asterisco y cierra con asterisco barra */
;	fin de línea
entero a	definición de variable entera
boolean a	definición de variable lógica
leer (a)	el valor de a es asignado por una interface entrada de datos (ej. teclado)
escribir("El valor de a es ",a)	mostrar una salida (ej. pantalla)

a = b	asigna a la variable a el valor de b
cont = cont + 1	incrementa a la variable cont en 1 (contador)
acum=acum+valor	incrementa acum en valor (acumulador)
SI condicion ……. ……. FINSI	si cumple la condición realizar instrucciones hasta FINSI
SI condicion ……. ……. SINO ……. ……. FINSI	si cumple la condición realizar instrucciones hasta SINO si no cumple la condición realizar las instrucciones desde SINO hasta FINSI
MIENTRAS condicion ……. ……. FINM	mientras cumpla la condición realizar las instrucciones hasta FINM
HACER …….	ejecuta las instrucciones y mientras cumpla la condición volverá a ejecutar las instrucciones

…… MIENTRAS condicion	
PARA i=0 HASTA 9, Inc +1 …… …… FINPARA	realizar las instrucciones 10 veces, ya que $0 \leq i \leq 9$, Inc +1 indica que i se incrementara en 1
(a==b)	condición si a y b son iguales
$(a \neq b)$	condición si a y b son distintos
$(a \geq b)$	condición si a es mayor o igual que b
(a AND b)	condición si se cumple a y b
(a OR b)	condición si se cumple a o b (una o las dos)
(NOT b)	condición si se cumple la negación de b
mod	congruencia módulo ejemplos: 8 mod 3 es 2 (2 es el resto de la división de 8 por 3); 16 mod 2 es 0 (0 es el resto de la división de 16 por 2)

Primer ejemplo en C:

Ahora, veamos cómo se haría el primer ejemplo en C, para que notemos la semejanza entre el pseudocódigo y un lenguaje de programación propiamente dicho. Les proponemos que tanto el martes como miércoles traten de hacerlo funcionar y generen el cálculo del área de un rectángulo de una forma semejante.

En caso de tener problemas con los primeros pasos para instalar el CodeBlock e iniciar los primeros ejemplos en C, recomendamos ir al final del documento donde explicamos la instalación.

```
/* Programa: Suma de dos números  */
/* Pre Condición: Saber sumar  */
/* Post Condición: Conseguir la suma de dos números cualquiera  */

#include <conio.h>
#include <stdio.h>

int main()
{
   int n1, n2;

   printf( "\n   Introduzca el primer número (entero): " );
   scanf( "%d", &n1 );

   printf( "\n   Introduzca el segundo numero (entero): " );
   scanf( "%d", &n2 );

   getch(); /* Pausa */

   return 0;
```

```
}
```

¿Qué opinan? ¿Se parecen entre si el pseudocódigo y C? Los invitamos a resolver una multiplicación y calcular un área.

Ejemplo, distintos tipos de datos:

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

int main()
{

    //Recuerden que es muy importante comentar el codigo, para que en el futuro
    //tanto ustedes como 3ros sepan que es lo que hicieron o trataron de hacer

    //El PRINTF es el primer comando que vamos a utilizar, nos sirve para mostrar
    //por la pantala lo que querramos...
    printf("Bienvenidos a Programacion de computadoras!!! \n");

    ////////////////////TEORIA//////////////////////////////////////////////////////////////////
```

```
    //En la primer practica trataremos de hacer operaciones basicas con numeros
    //incresados por el teclado, vamos a trabajar con 5 numeros. Enteros, flotantes, dobles y largos.

    //Cual es la diferencia entre estos TIPOS DE DATOS???
    //los int van desde el -32768 al 32767
    //los long -2147483648 a 2147483647
    //los doubles de  1.7E-308 a 1.7E+308
    //los long de 1.7E-308 a 1.7E+308   ó 3.4E-4932 a 1.1E+4932
    ////////////////////FIN_TEORIA//////////////////////////////////////////////////////////////////////

    //¡¡¡¡¡Definimos los datos que vamos a usar!!!!!!!
    int numero1;
    int numero2;
    float numero3;
    double numero4;
    long numero5;

    //Pedimos los ingresos por pantalla:
    printf ("Ingrese el valor del primer entero: \n" );
    //Scanf guarda en la memoria el valor ingresado, dicho valor se guardara en la variable dicha luego del &
```

```
    //Es necesario "avisar" al scanf que tipo de dato es el que se guardara...
    scanf( "%d", &numero1 );

    //Repetimos lo mismo con los otros numeros
    printf ("Ingrese el valor del 2do entero: \n");
    scanf( "%d", &numero2 );

    printf ("Ingrese el valor del  flotante: \n");
    scanf( "%f", &numero3 );

    printf ("Ingrese el valor del  double: \n");
    scanf( "%lf", &numero4 );

    printf ("Ingrese el valor del  long: \n");
    scanf( "%f", &numero5 );

    //Hagamos algunas operaciones con los valores ingresados:
    float suma1 = numero1 + numero3;
    float suma2 = numero1 + numero2;
    float producto1 = numero4*numero5;
    float producto2 = numero1*numero1;
    float division = suma1/producto1;
```

```
    //Mostremos por pantalla los resultados
    printf ("La primer suma da:  %.2f \n", suma1);
    printf ("La segunda suma da:  %.2f \n", suma2);
    printf ("El primer producto da:  %.3f \n", producto1);
    printf ("EL segunda producto da:  %.2f \n", producto2);
    printf ("La division da:  %.1f \n", division);


    printf ("------------------------- \n\n");
    printf ("Con esto terminamos la primer semana introductoria de programacion \n\n");
    printf ("1)Para que sirve el &.2 o &.3 al mostrar un flotante???\n");
    printf ("2)Como haria para ROMPER este codigo, que error seria???\n");
    printf ("3)Una vez entendido todo lo anterior, trate de mejorar el codigo, y crear potenciaciones y radicaciones\n");
    printf ("--------------------------");
    return 0;
}
```

Operadores relacionales y operadores lógicos en c

Si bien ya vimos los operadores lógicos AND, OR, ==, <=, etc; veamos cómo se utilizan en C con algunos ejemplos.

Se llaman operadores relacionales o de comparación a aquellos que permiten comparar dos valores evaluando si se relacionan cumpliendo el ser menor uno que otro, mayor uno que otro, igual uno que otro, etc. Los operadores lógicos permiten además introducir nexos entre condiciones como “y se cumple también que” ó . “o se cumple que”.

Los operadores de comparación o relacionales básicos en C son:

Operador	**Significado**
Operador <	Menor que
Operador <=	Menor o igual que
Operador >	Mayor que
Operador >=	Mayor o igual que
Operador ==	Igual a
Operador !=	Distinto de ó no igual que

Es importante tener en cuenta que para comparar si una variable A es igual a otra debemos usar A == B en lugar de A = B. El doble signo igual se usa en comparaciones mientras que el signo igual sencillo se usa en asignaciones.

Para determinar si una variable A tiene distinto contenido que una variable B debemos usar A != B. En C no se admite la sintaxis A <> B que usamos en pseudocódigo.

Suponiendo que la variable A tiene un valor A = 5 y la variable B un valor B = 7 podríamos plantear las siguientes expresiones:

Expresión	**Resultado**	**Ejemplo código**
A == B	0 (falso)	printf ("A == B vale %d\n", (A == B));

A != B	1 (verdadero)	printf ("A != B vale %d\n", (A != B));
A > B	0 (falso)	printf ("A > B vale %d\n", (A > B));
A >= B	0 (falso)	printf ("A >= B vale %d\n", (A >= B));
A < B	1 (verdadero)	printf ("A < B vale %d\n", (A < B));
A <= B	1 (verdadero)	printf ("A <= B vale %d\n", (A <= B));

Hay que tener en cuenta que en C al no tener el tipo booleano entre sus tipos predefinidos el resultado de evaluar estas expresiones es un cero ó un uno, equivaliendo el cero a “falso” y el uno a “verdadero”.

Ejecuta un programa definiendo las variables A y B como de tipo entero, asígnales los valores correspondientes y comprueba las expresiones usando el código de ejemplo que hemos indicado. Define otras variables y haz algunas comprobaciones por tu cuenta.

Operadores lógicos en c

Los operadores lógicos básicos en C son los siguientes:

Operador	**Significado**
Operador &&	Operador lógico and
Operador ||	Operador lógico or
Operador !	Operador lógico not

Suponiendo que tenemos cuatro variables A, B, C y D cuyos valores se han establecido en A = 5, B = 7, C = 2, D = 5 podríamos evaluar estas expresiones:

Expresión	Pregunta equivalente	Resultado	Ejemplo código
(A == B) && (A < B)	¿Es A igual a B y A menor que C?	0 (falso)	printf ("Pregunta (A == B) && (A < B) vale %d\n", (A == B) && (A < B));
(A == 5) || (A > 7)	¿Es A igual a 5 ó es A mayor que 7?	1 (verdadero)	printf ("Pregunta (A == 5) || (A > 7) vale %d\n", (A == 5) || (A > 7));
! (A == 5)	¿A es NO igual a 5?	0 (falso)	printf ("Pregunta ! (A == 5) vale %d\n", ! (A == 5));

Práctica Unidad 1:

1- Dadas las variables de tipo entero con valores A = 5, B = 3, C = -12 indicar si la evaluación de estas expresiones daría como resultado verdadero o falso:

a) A > 3 **b) A > C** **c) A < C**

d) B < C **e) B != C** **f) A == 3**

g) A * B == 15 **h) A * B == -30** **i) C / B < A**

j) C / B == -10 **k) C / B == -4** **l) A + B + C == 5**

m) (A+B == 8) && (A-B == 2) **n) (A+B == 8) || (A-B == 6)**

o) A > 3 && B > 3 && C < 3 **p) A > 3 && B >= 3 && C < -3**

2- Realizar la prueba de escritorio y tablas de verdad para los ejercicios m,o y p.

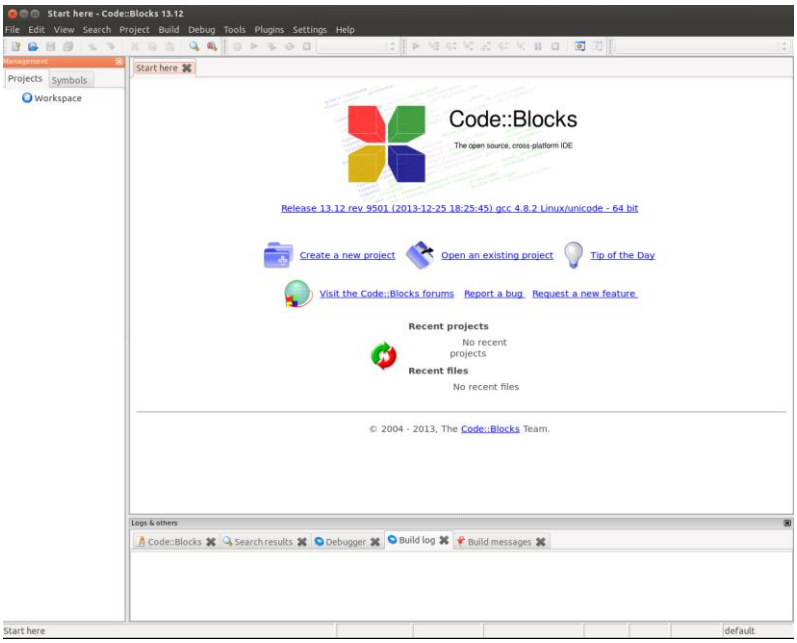
Anexo: ***Programación de Computadoras - Instalación IDE Code::Blocks***

Para realizar los trabajos prácticos de la cursada emplearemos el lenguaje de programación C. Éste es un lenguaje de alto nivel y de propósito general desarrollado por Brian Kernighan y Dennis Ritchie en los laboratorios Bell, para utilizarse en la programación del sistema operativo UNIX.

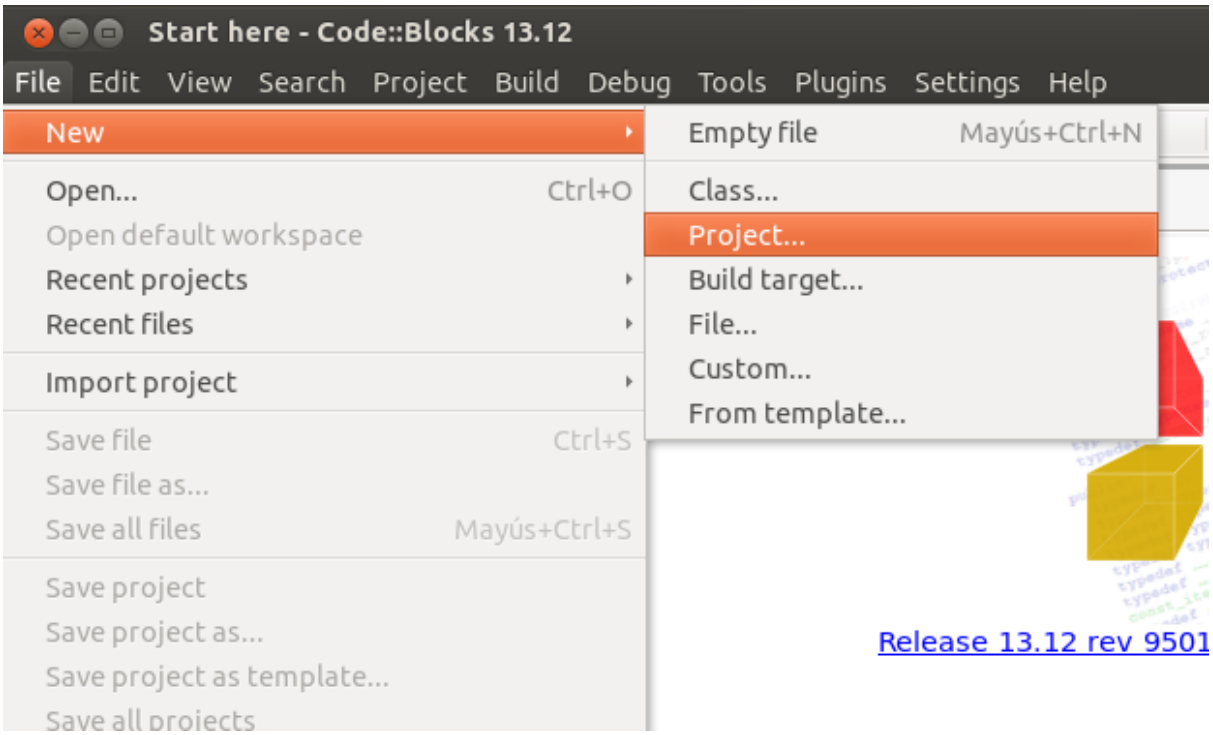
Un programa C se escribe en algún procesador de texto y se salva en un archivo con extensión C. Ese archivo se utiliza como entrada de un compilador, cuya salida será un archivo ejecutable para la plataforma a la que esté apuntado dicho compilador. Así, por ejemplo en Windows, el compilados MinGW nos dejará un archivo exe que podremos ejecutar directamente (haciendo doble click sobre el mismo, por ejemplo). Si deseamos depurar el programa (esto es correrlo en un modo especial para que podamos ver el estado del programa cuando se produce un error), deberemos utilizar un depurador al efecto. Este proceso es engorroso de realizar a mano, y para poder trabajar con comodidad se emplea lo que se conoce como IDE (Integrated Developenment Environment, Entorno de Desarrollo Integrado). Una IDE es una aplicación que integra facilidades de edición, compilación y depuración de programas.

A fin de realizar todos los ejercicios de la cursada utilizaremos la IDE Code::Blocks.

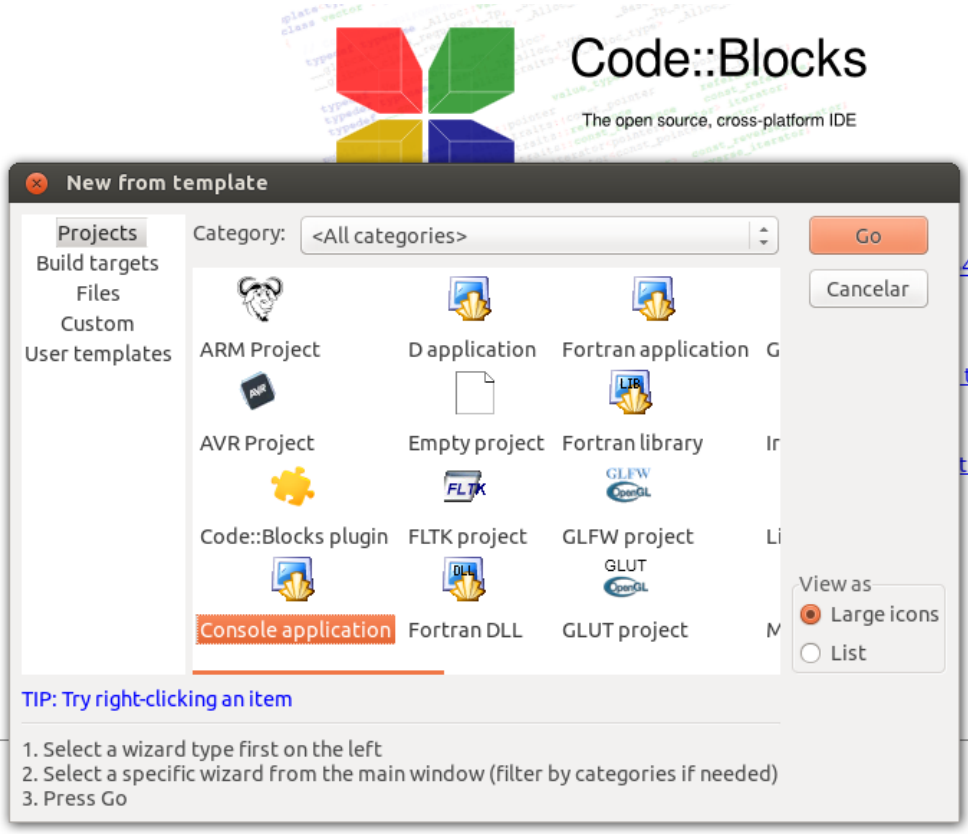
Para descargarla, nos dirigiremos al link http://www.codeblocks.org/downloads/26 y procederemos a descargar la versión más adecuada para nuestro sistema operativo. Para Windows, es conveniente descargar la versión codeblocks-17.12mingw-setup.exe, debido a que dicho sistema operativo carece de un compilador C y esta versión de Code::Blocks lo trae incorporado. Una vez instalada procedemos a abrirla. Nos encontraremos con una ventana como la siguiente:



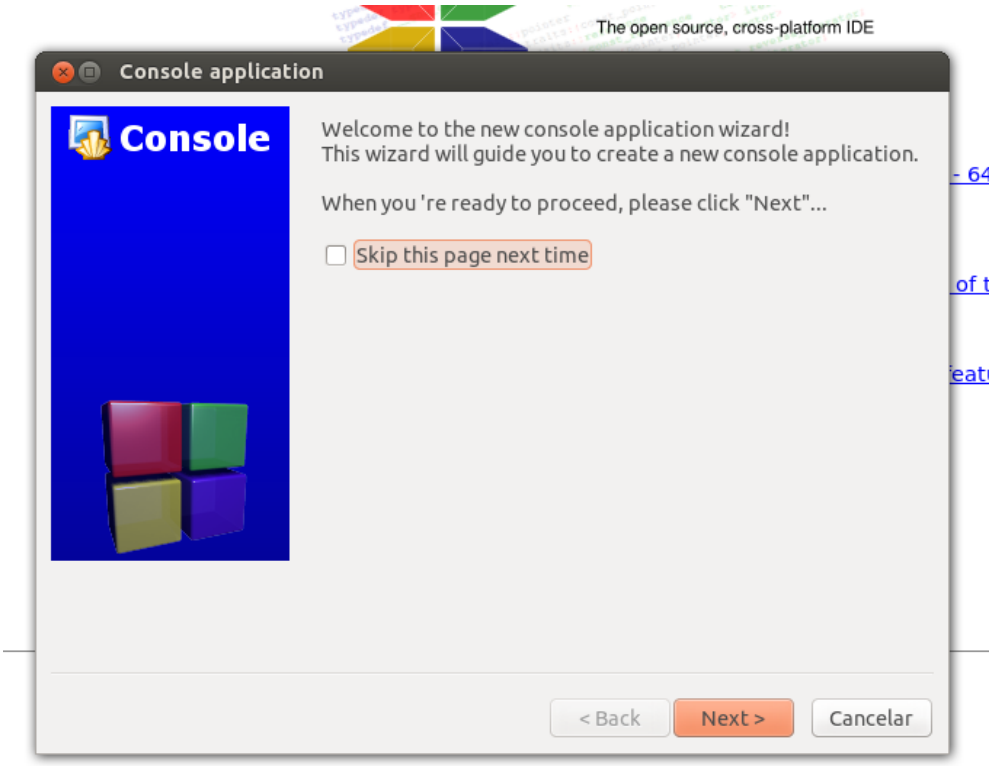
Todos los ejemplos y trabajos prácticos estarán contenidos en un proyecto de Code::Blocks. Para crearlo, iremos al menú File / New / Project



En la ventana siguiente, elegiremos “Console application” y haremos click en el botón “Go”

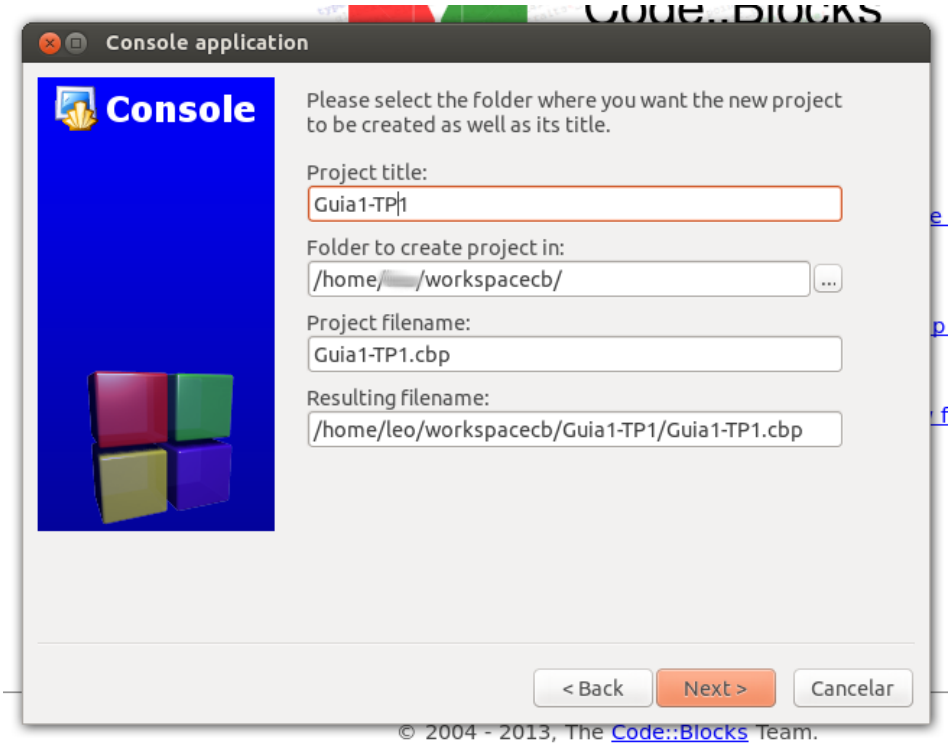


Hacemos click en el botón “Next” de la ventana siguiente:

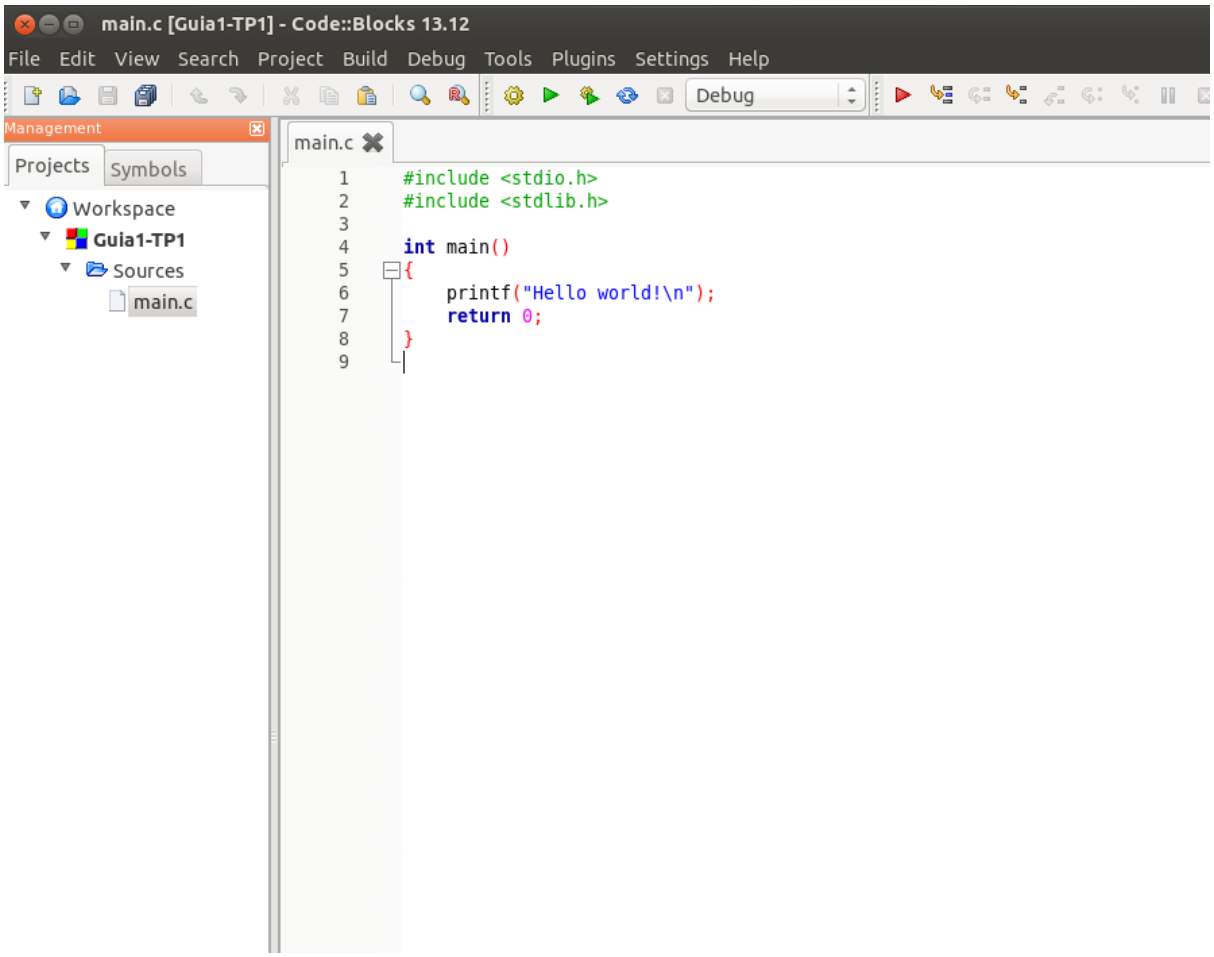


Seleccionamos el lenguaje C y hacemos click en el botón “Next”:

Indicamos el nombre del proyecto y la carpeta donde se guardará. En este ejemplo se trata de un sistema Linux, en Windows podría ser c:\workspacecb o cualquier otro nombre.



Damos “Finish” sin modificar nada en la pantalla siguiente y Code::Blocks debería haber creado una estructura de proyecto como como la siguiente:



En dicha estructura de proyecto, podemos ver lo que se denomina “Workspace” o lugar de trabajo. Representa al lugar en que se se almacenan nuestros proyectos y corresponde a la carpeta de trabajo que indicamos al crear el proyecto. Dentro del mismo encontramos la definición del proyecto (Guia1-TP1) y dentro del mismo una carpeta denominada “Sources (“fuentes” en español). Dentro de la misma se almacenan todos los archivos del proyecto que contengan código fuente.

Cuando el procesador de una computadora ejecuta un programa, lo hace leyendo desde memoria instrucciones de un código que entiende el procesador. Este código es de lo que denominamos bajo nivel y está compuesto por instrucciones básicas de movimiento de datos en la memoria, o entre ésta y el procesador y las operaciones de las que es capaz este último: sumas, restas, desplazamiento de bits y algunas más. A un programa en este formato (que comprende el procesador) se lo denomina código objeto, porque es objeto de la ejecución del procesador. Si bien es posible escribir un programa utilizando estas instrucciones básicas (y a veces se sigue haciendo , debido a

circunstancias especiales como ser la necesidad de optimizar la velocidad de ejecución o el espacio que el programa ocupa en memoria) a partir de cierto tamaño la tarea se vuelve difícil, proclive a errores y tediosa.

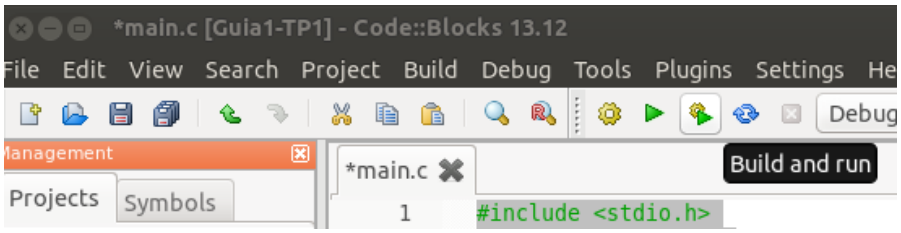
A partir del reconocimiento de que hay secuencias de operaciones que se escriben en un programa una y otra vez (por ejemplo, tomar un valor de memoria, sumarle otro y volver a guardarlo en la misma posición o tomar una serie de caracteres y mostrarla por pantalla) se ideó una nueva forma de programar: en lugar de hacerlo directamente en el código del procesador (también denominado "código máquina", "assembly" o ensamblador) se creó un nuevo lenguaje, que fuera más amigable para los humanos y que permitiera expresar mejor las soluciones diseñadas que el ensamblador. A este nuevo código se lo denominó fuente (source en inglés) porque es fuente u origen del código objeto y decimos que es de "alto nivel". Todo aquello que esté más cerca del programador y más lejos del procesador se denomina de alto nivel y viceversa. Como el procesador no comprende este nuevo código, éste debe procesarse por medio de un programa que lo "traduzca" en el código que sí comprende el procesador. Hay dos formas de hacer esto: a medida que el programa se ejecuta (decir que el programa se ejecuta es decir que se lee instrucción por instrucción y el procesador ejecuta las acciones indicadas en las mismas) o se transforma de una vez al principio y luego se ejecuta el código objeto resultante. Estas dos operaciones se denominan interpretar y compilar, respectivamente. Un intérprete lee el código fuente línea por línea, traduce (interpreta) la línea recién leída y se la pasa al procesador para que la ejecute. Un compilador lee el código del programa completo y devuelve una versión del mismo pero en código objeto, que ejecuta directamente el procesador. La operación de compilar se realiza una sola vez, antes de ejecutar el programa, mientras que la de interpretar se realiza cada vez que se ejecuta. Esto redunda en una menor eficiencia en la ejecución (se ejecuta más lentamente, debido a que el intérprete agrega una tarea más al procesador). Decimos que en el caso del intérprete, la traducción se realiza en **tiempo de ejecución** mientras en en el compilador se realiza en **tiempo de compilación**. En el caso de C, se trata de un lenguaje compilado, y esto es así por razones de eficiencia.

Volviendo al motivo de nuestro trabajo, dentro de la carpeta sources encontramos un archivo denominado "main.c". Todos los archivos de código fuente C tienen extensión .c. Cuando se inicie la ejecución de un programa C, siempre se lo hará por la función denominada "main" ("principal" en inglés). El archivo main.c contiene

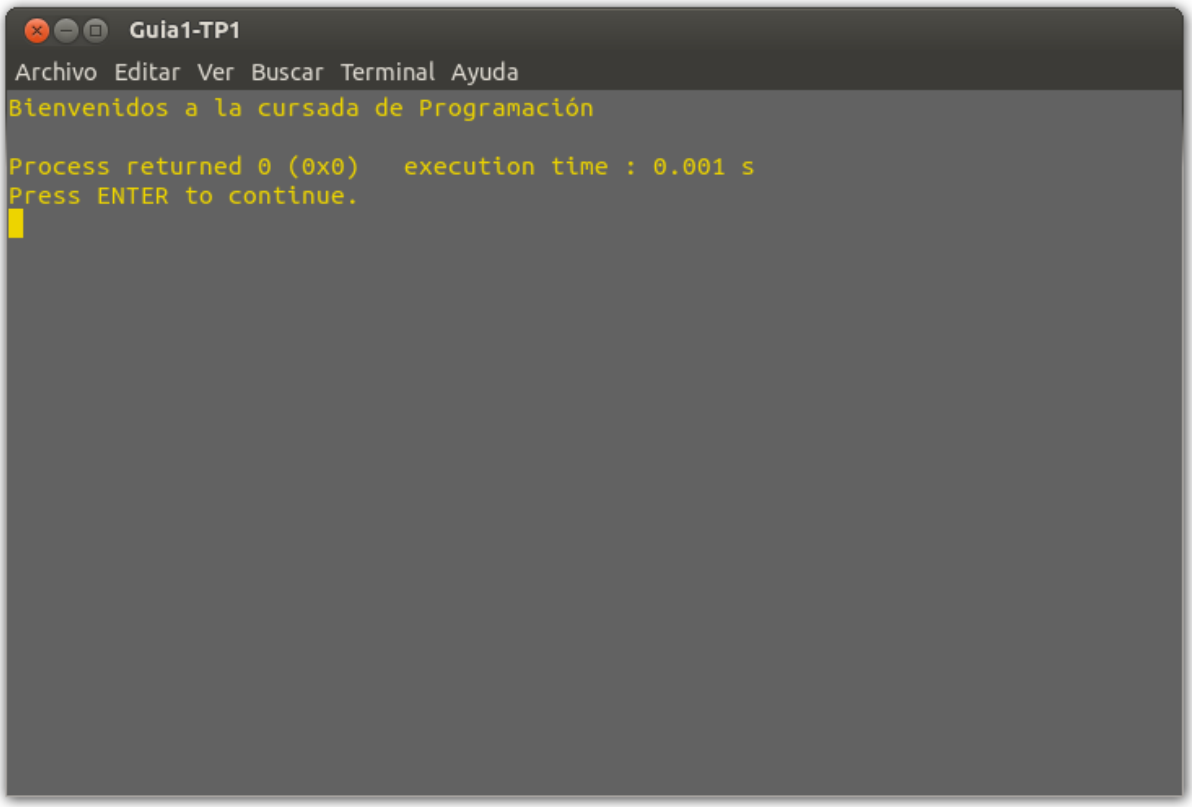
(podemos verlo si hacemos doble click sobre el mismo) una función denominada main, creada por Code::Blocks. Vamos a hacerle un cambio, para que quede de la siguiente manera:

```
#include <stdio.h>

#include <stdlib.h>


int main()

{

    printf("Bienvenidos a la cursada de Programación\n");

    return 0;

}
```

Así como está, es posible ejecutar este programa. Para ello, damos click sobre el botón “Build and run” (construir y ejecutar) de la barra de tareas:



La IDE debería respondernos con una ventana de consola similar a la siguiente:



Damos enter para cerrarla. En general el flujo de trabajo será siempre similar: Creamos un proyecto, introducimos el código fuente de nuestra solución en el mismo, compilamos y corremos para probar que funcione, corregiremos los errores que pudieran surgir y volvemos a compilar y correr hasta que no surjan nuevos errores.

Vamos a mirar nuestra función main con un poco más de detalle. Empezamos con lo que se denomina “includes”, inclusión de librerías en el código actual (to include es incluir en inglés):

```
#include <stdio.h>

#include <stdlib.h>
```

Cuando se escriben programas, pronto se nota que algunas tareas se realizan en

todos los programas. Por ejemplo, mostrar un mensaje por consola es una de las mismas. El lenguaje C ofrece una librería estándar (una librería es un conjunto de funciones agrupadas porque realizan tareas relacionadas) con estas funciones ya definidas y disponibles para usar. En este caso se incluyen la librería estándar stdlib.h (funciones de uso común en todos los programas) y stdio.h (funciones de entrada y salida)

Lo que sigue es la **declaración** de la función main:

```
int main()
{
```

aquí se define el tipo del valor que retorna la función (int: entero, más adelante veremos todos los tipos de datos que ofrece C) , el nombre de la misma y, entre paréntesis, los tipos y nombres de los valores que la función recibe como parámetros de entrada (en nuestro caso, main no recibe nada y por lo tanto no hay nada entre los paréntesis). Las funciones en C (y en cualquier otro lenguaje de programación) en general reciben datos de entrada, realizan algún proceso con ellos y devuelven algún resultado. Puede ocurrir que la función no reciba ningún dato de entrada, como main o puede ocurrir que no devuelva ningún dato, en cuyo caso genera un efecto colateral (como printf, que recibe un texto a mostrar y lo hace por consola, sin devolver nada) y también puede ocurrir que no reciba ni devuelva y genere un efecto colateral.

La llave que puede verse debajo de la declaración de main, define el principio del cuerpo de la función. A partir de la misma, y hasta la llave que la cierre, están las instrucciones que deben ejecutarse cuando se invoque a la función main.

Dentro del cuerpo de main, vemos las dos únicas instrucciones que contiene:

printf("Bienvenidos a la cursada de Programación\n");

Esta instrucción saca por consola el texto que recibe como parámetro. Más adelante la veremos con más detalle. la siguiente:

return 0;

devuelve el valor 0 al llamador, en este caso, Code::Blocks. En el caso de main, este valor se utiliza para informar al sistema operativo si el programa terminó normalmente o con algún problema. 0 significa que se terminó normalmente. Más adelante profundizaremos los valores de retorno y los parámetros de las funciones.