|  |  |
| --- | --- |
| **Depto. Desarrollo Productivo y Tecnológico**  ***Licenciatura en Sistemas***  ***Programación de Computadoras***  ***Docentes y Contacto:***  Jorge Golfieri – Natalia Romero – Nicolás Perez  [jgolfieri@hotmail.com](mailto:jgolfieri@hotmail.com)  [nataliab\_romero@yahoo.com.ar](mailto:nataliab_romero@yahoo.com.ar)  [nperez\_dcao\_smn@outlook.com](mailto:nperez_dcao_smn@outlook.com) |  |

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

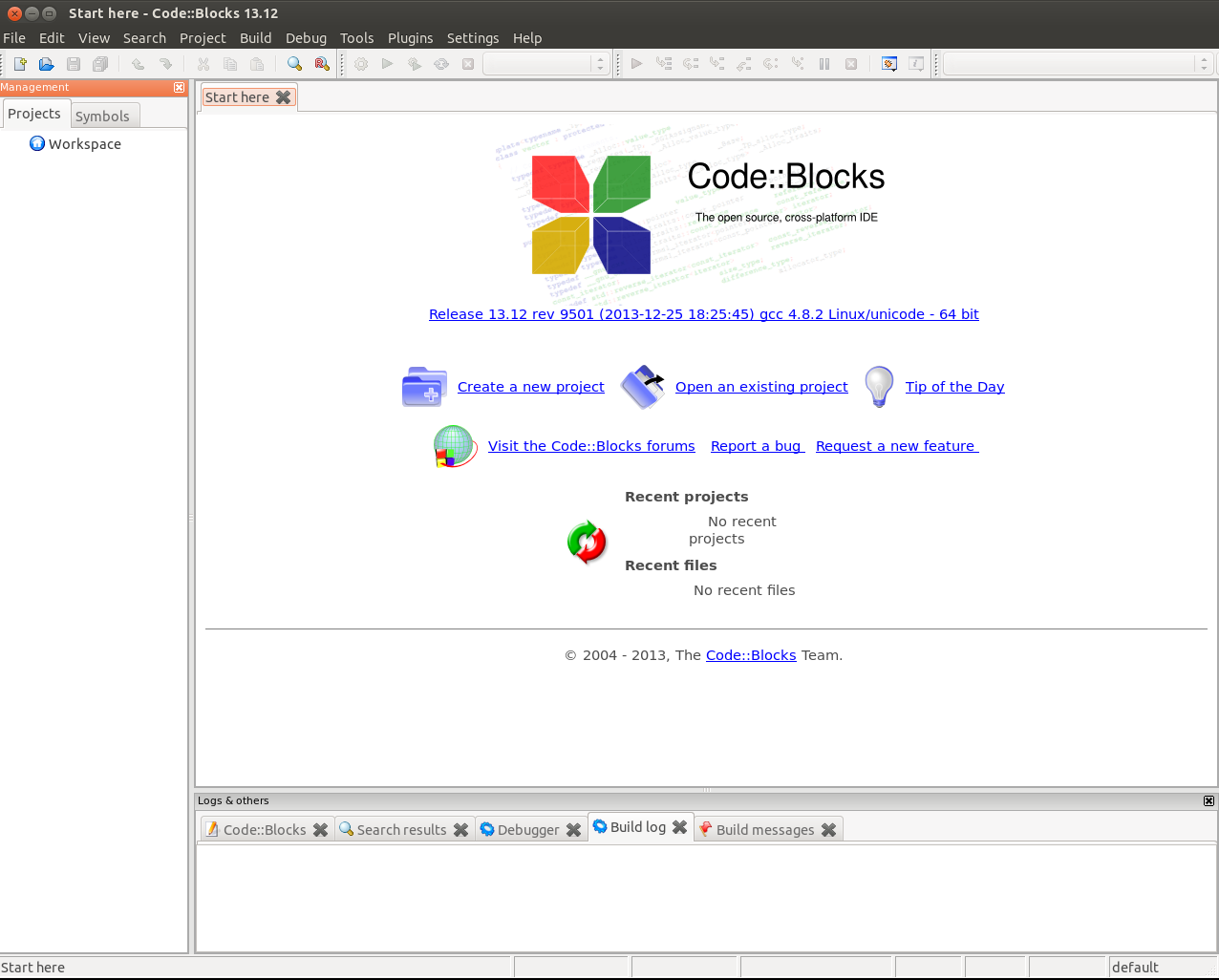
Para realizar los trabajos prácticos de la cursada emplearemos el lenguaje

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.

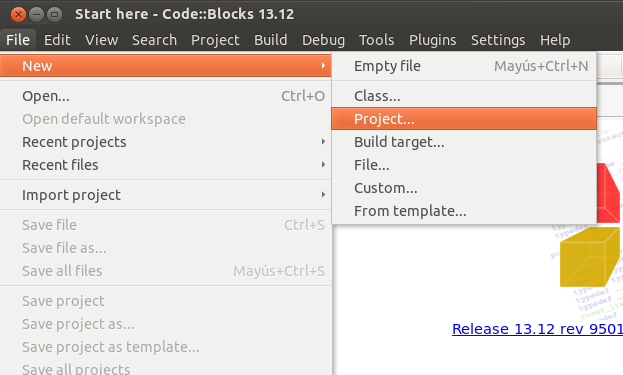
Un programa C se escribe en algún procesador de texto y se salva en un archivo con extensión C. Ese archivo se utiliza como entrada de un compilador, cuya salida será un archivo ejecutable para la plataforma a la que esté apuntado dicho compilador. Así, por ejemplo en Windows, el compilados MinGW nos 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.

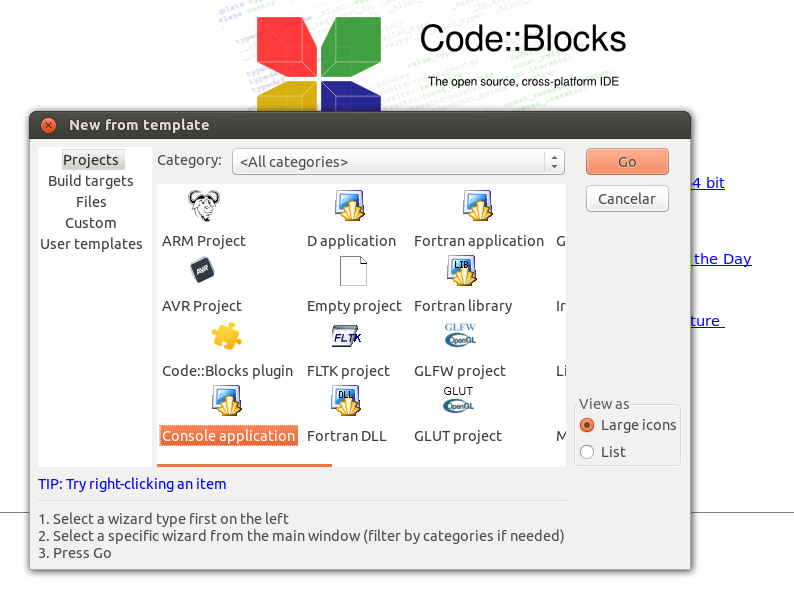
Para descargarla, nos dirigiremos al link   
<http://www.codeblocks.org/downloads/26> y procederemos a descargar la versión más adecuada para nuestro sistema operativo. Para Windows, es conveniente descargar la versión codeblocks-17.12mingw-setup.exe, debido a que dicho sistema operativo carece de un 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:



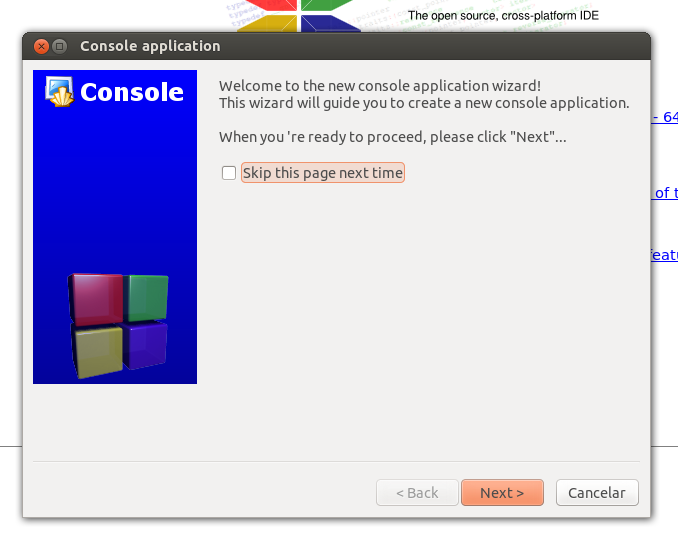
Todos los ejemplos y trabajos prácticos estarán contenidos en un proyecto de Code::Blocks. Para crearlo, iremos al menú File / New / Project



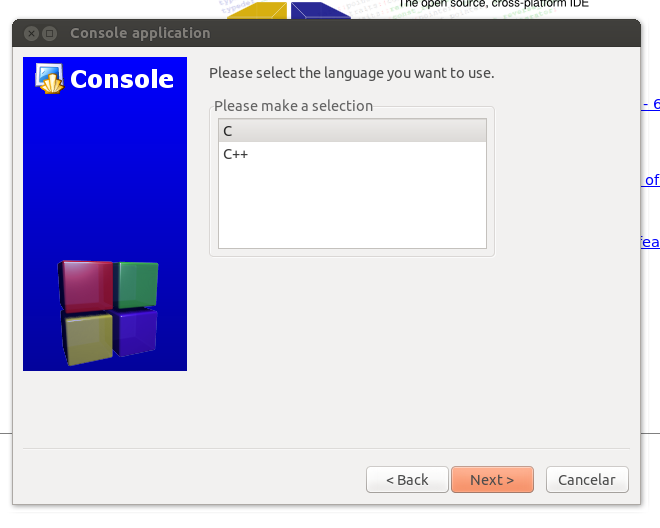
En la ventana siguiente, elegiremos “Console application” y haremos click en el botón “Go”



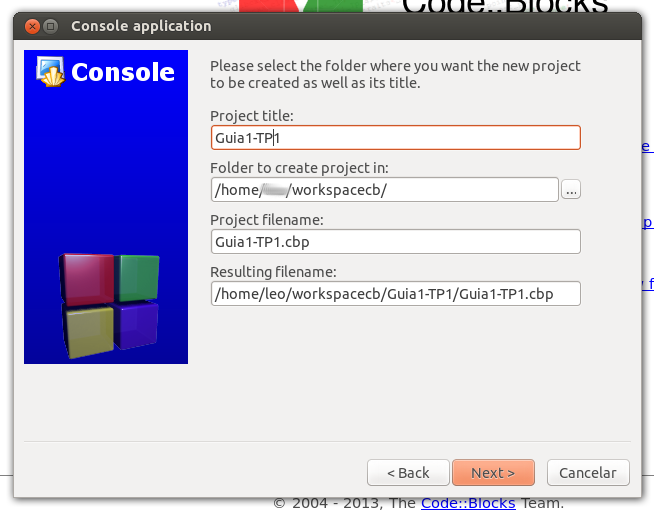
Hacemos click en el botón “Next” de la ventana siguiente:



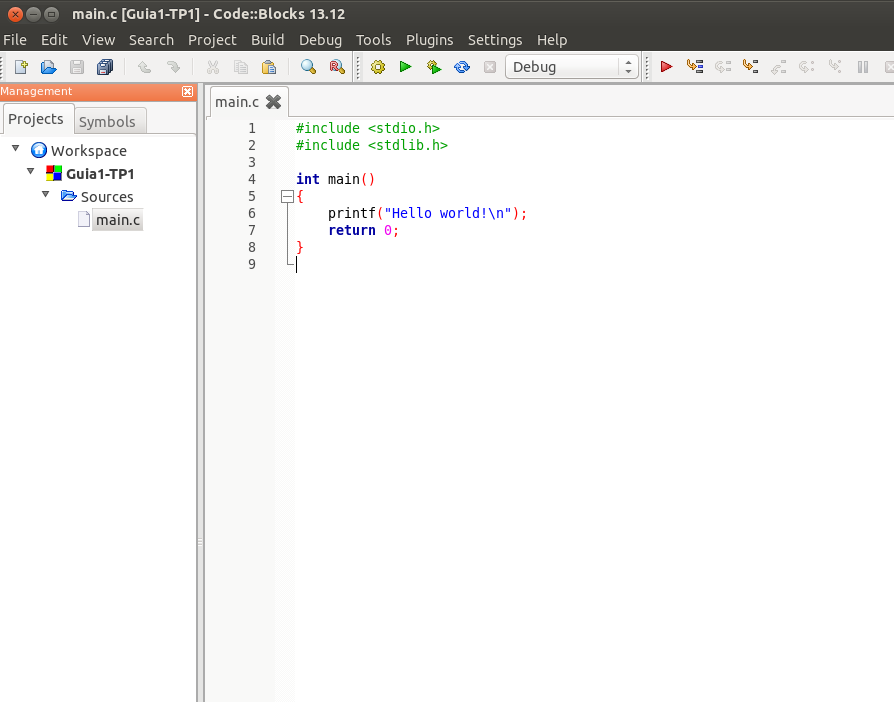
Seleccionamos el lenguaje C y hacemos click en el botón “Next”:



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()

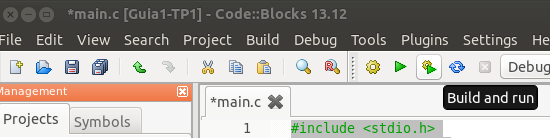
{

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

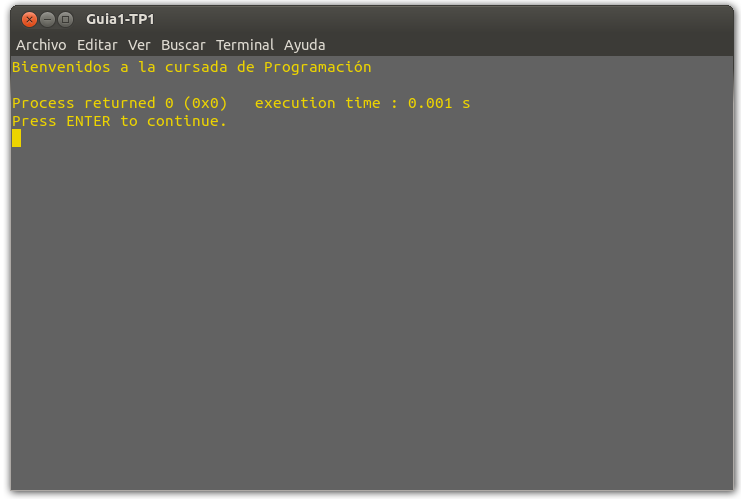
return 0;

}

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.