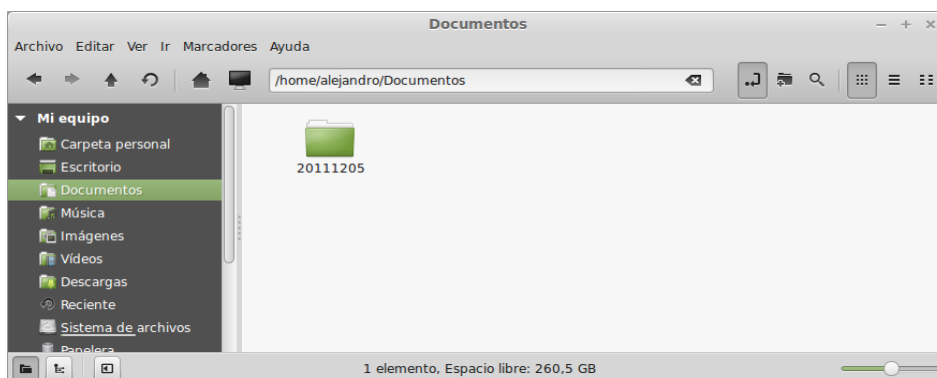


PONTIFICIA UNIVERSIDAD CATOLICA DEL PERU
FACULTAD DE CIENCIAS E INGENIERIA

INGENIERIA INFORMATICA
Laboratorio Preliminar 0B

Preparación

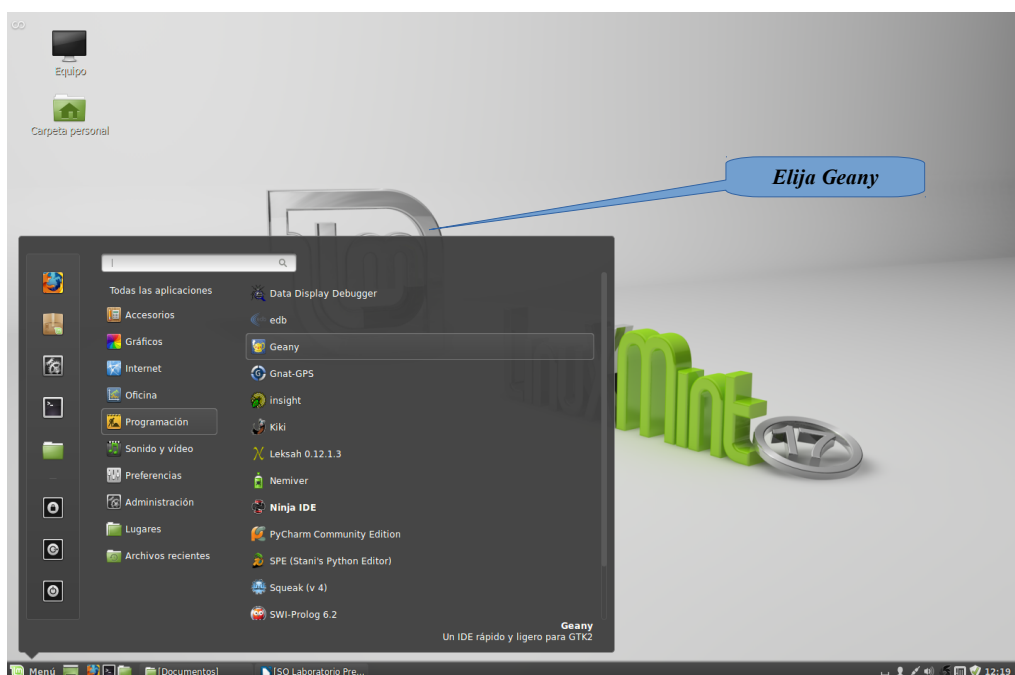
Con la ayuda del navegador de archivos (en **Linux Mint** es conocido como *Nemo*) ubíquese en la carpeta *Documentos* y cree una carpeta que tenga como nombre su código de alumno (por ejemplo 20111205). Tal como se muestra a continuación:



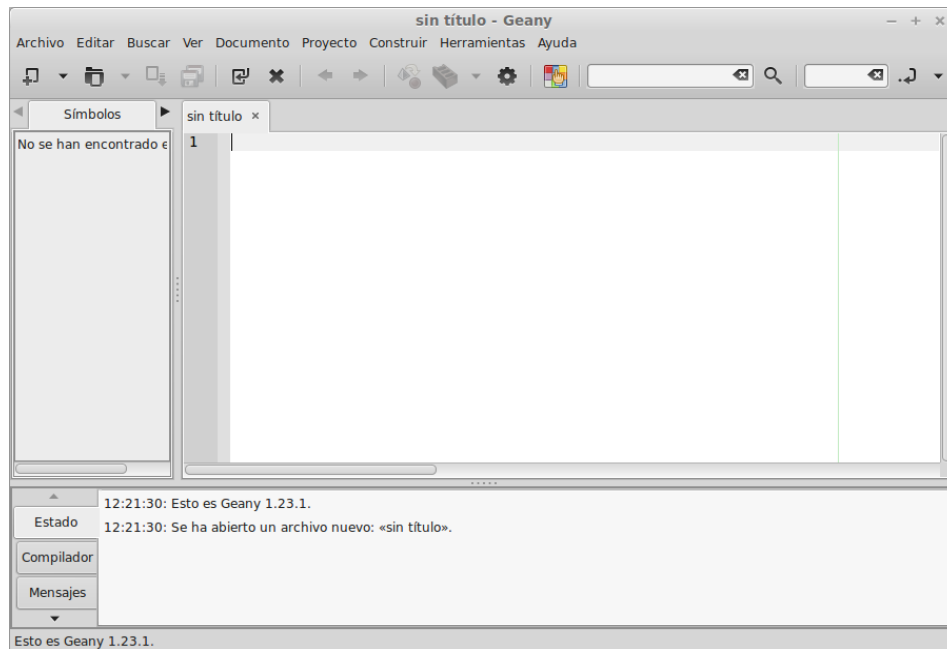
El objetivo es colocar en este último directorio creado, todos los archivos que vamos a escribir en esta sesión de trabajo. Luego comprimiremos esta carpeta y la colocaremos en Intranet.

Nuestros primeros programas – Identificadores de procesos y de usuario.

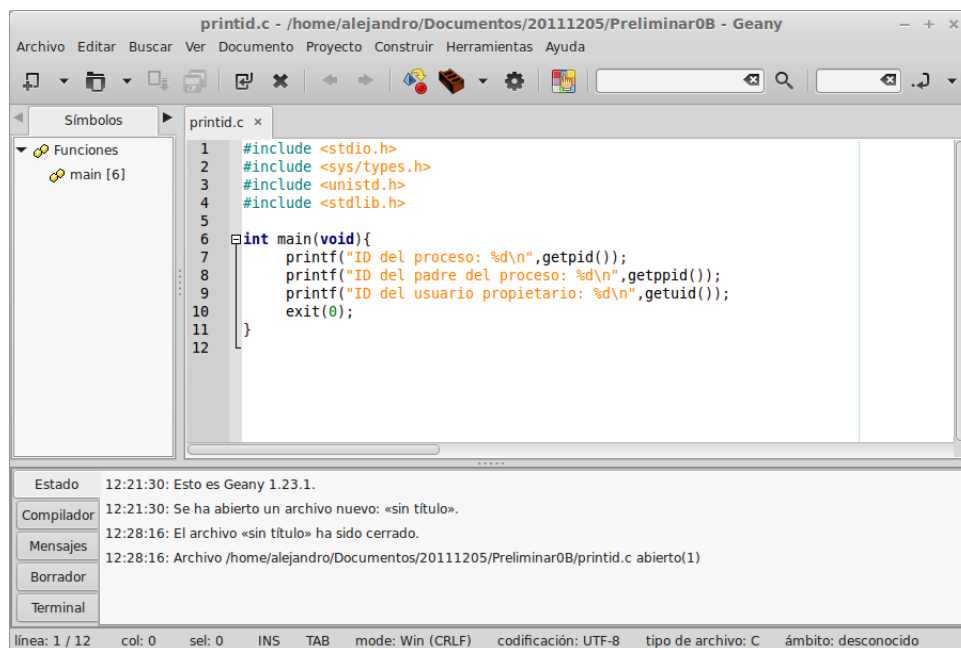
Existe muchos editores de texto que pueden ser empleados para escribir código, el que se ha elegido es un editor bastante liviano y sencillo, su nombre es *Geany*. Elija del menú de Linux Mint el editor, tal como se muestra a continuación:



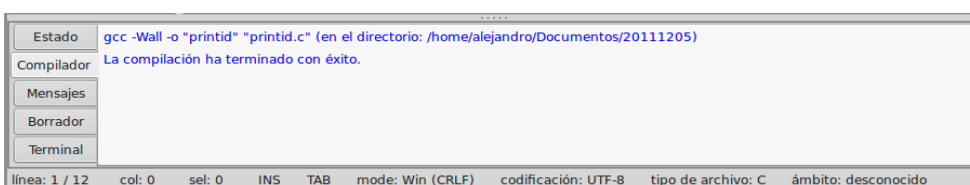
Obtendrá la siguiente ventana:



1.- A continuación escriba el siguiente programa fuente con nombre *printid.c* y grabe (*Archivo/Guardar como*) el archivo en el directorio */home/alulab/Documentos/20111205*

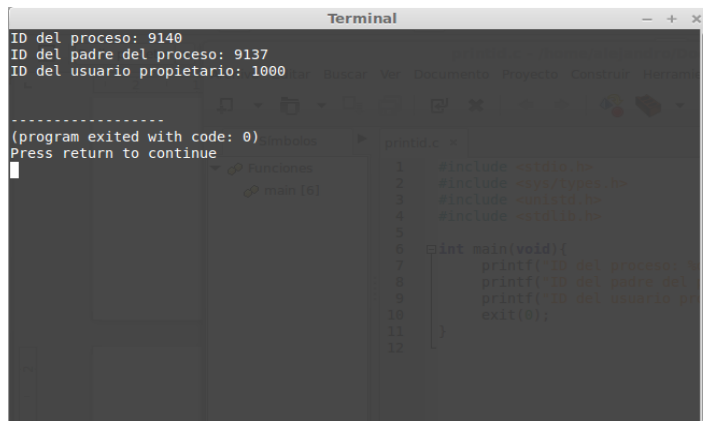


2. Desde el menú de Geany compile (*Construir/Construir*) el programa. También puede emplear la tecla F9 para compilarlo.



3. Desde el menú de Geany ejecute (*Construir/Ejecutar*) el programa. También puede emplear la tecla F5 para ejecutarlo.

La salida se mostrará en una terminal aparte, tal como se muestra a continuación:

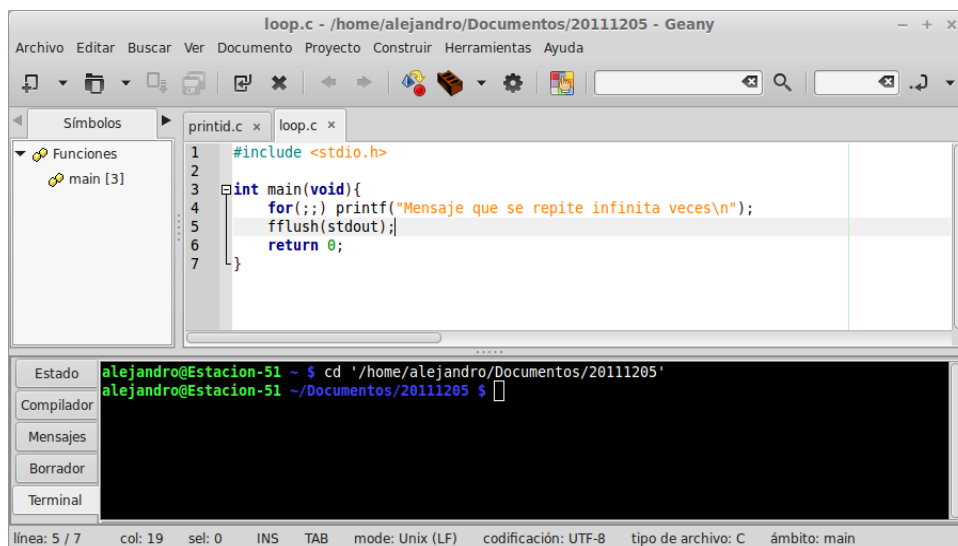


```

Terminal
ID del proceso: 9140
ID del padre del proceso: 9137
ID del usuario propietario: 1000

(program exited with code: 0)
Press return to continue
  
```

4. Escriba este segundo programa y grábelo con nombre *loop.c*, luego compile y ejecute dicho programa desde la terminal de Geany. Para lograr esto haga *click* en la pestaña *Terminal* en la parte inferior izquierda de Geany.



```

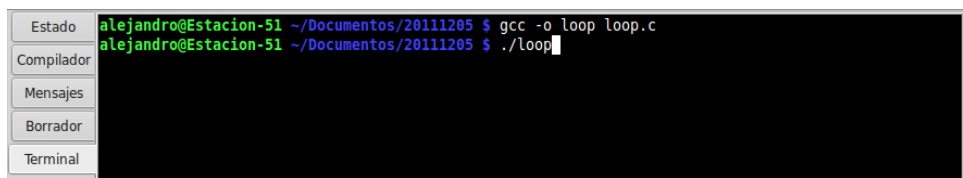
loop.c - /home/alejandro/Documentos/20111205 - Geany
Archivo  Editar  Buscar  Ver  Documento  Proyecto  Construir  Herramientas  Ayuda

Símbolos  printid.c x  loop.c x
  Funciones
    main [3]

1  #include <stdio.h>
2
3  int main(void){
4      for(;;) printf("Mensaje que se repite infinita veces\n");
5      fflush(stdout);
6      return 0;
7  }

Estado  alejandro@Estacion-51 ~ $ cd '/home/alejandro/Documentos/20111205'
Compilador  alejandro@Estacion-51 ~/Documentos/20111205 $ 
Mensajes
Borrador
Terminal
línea: 5 / 7  col: 19  sel: 0  INS  TAB  mode: Unix (LF)  codificación: UTF-8  tipo de archivo: C  ámbito: main
  
```

A continuación escriba los siguientes comandos para compilar y ejecutar el programa:



```

Estado  alejandro@Estacion-51 ~/Documentos/20111205 $ gcc -o loop loop.c
Compilador  alejandro@Estacion-51 ~/Documentos/20111205 $ ./loop
Mensajes
Borrador
Terminal
  
```

Como ha de esperarse se produce la salida interminable de mensajes. Presione Ctrl+C para detener la salida. Lo que en ese momento sucede es que al presionar Ctrl+C, el sistema envía una señal de termino al proceso, quien al recibirla termina.

Un proceso se encuentra ejecutándose en **primer plano** (*foreground*) si interactúa con el usuario recibiendo las entradas del teclado y enviando las salidas a la pantalla (mientras no se redirijan). En cambio un proceso se ejecuta en **segundo plano** (*background*) si no hace uso de la consola, dejándola libre para ejecutar algún otro comando. Lanzar un proceso en *background* es útil cuando éste puede

tomar mucho tiempo en ejecutarse. En símbolo “&” al final de un comando indica que estos se ejecutarán en segundo plano.

5. Modifique el programa *loop.c* eliminando la función `printf("Mensaje que se repite infinita veces\n")` (dejando el *for* vacío) Desde la terminal de Geany ejecute el programa *loop*, en segundo plano. Compruebe con el comando `ps` que *loop* se encuentra ejecutando.

6. Observe que cuando se ha lanzado el proceso en segundo plano, la línea de comandos ha quedado libre. Si usted presiona Ctrl+C no podrá detener el programa porque ya está desligada de la terminal. Usted tendrá que “matar” el proceso. Primero tendrá que averiguar el *pid* del proceso, para este fin emplee el comando `ps`, luego ejecute el siguiente comando:

```
kill -15 pid
```

donde *pid* es el número que corresponde al identificador del proceso el mismo que puede encontrarlo en la salida del comando `ps`.

7. Escriba, compile y ejecute los siguientes programas:

```

printid.c x loop.c x pfan.c x
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5 #include <sys/types.h>
6
7 /* Este programa crea procesos hijos en forma horizontal. */
8 /* Es decir el padre crea N hijos. */
9 /* Ejm 2.6 del libro UNIX Programacion Practica - Kay Robbins */
10 /* Steve Robbins */
11 /* Modificado por Alejandro Bello Ruiz - Informática PUCP */
12
13 #define N 8
14
15 int main(void)
16 {
17     int i,status;
18     pid_t child,pid_padre;
19
20     pid_padre=getpid();
21     for(i=0;i<N; ++i)
22         if((child=fork())<=0) break;
23     else fprintf(stderr,"Ciclo Nro %d \n",i);
24     fprintf(stderr,"Proceso con pid=%d y pid de padre= %d\n",getpid(),getppid());
25     if(pid_padre==getpid()) for(i=0;i<N; ++i) wait(&status);
26     return 0;
27 }
28
29
printid.c x loop.c x pfan.c x pchain.c x
1 #include <stdio.h>
2 #include <sys/types.h>
3 #include <unistd.h>
4 #include <sys/wait.h>
5
6 /* Este programa crea procesos hijos en forma vertical. Es decir */
7 /* el padre crea un hijo, este a su vez crea otro y asi en forma */
8 /* sucesiva. */
9 /* Ejm 2.5 del libro UNIX Programacion Practica - Kay Robbins */
10 /* Steve Robbins */
11 /* Modificado por Alejandro Bello Ruiz - Informática PUCP */
12
13 int main(void)
14 {
15     int i,status;
16     pid_t child; /* pid_t es un tipo definido en types.h */
17
18     for (i=1;i<4;++i) if((child=fork())) break;
19     fprintf(stderr,"Esta es la vuelta Nro %d\n",i);
20     fprintf(stderr,"Recibi de fork el valor de %d\n",child);
21     fprintf(stderr,"Soy el proceso %d con padre %d\n\n",getpid(),getppid());
22     wait(&status);
23     return 0;
24 }
25
26

```

```

1 #include <errno.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <string.h>
5 #include <unistd.h>
6 #include <sys/types.h>
7 int main(int argc, char *argv[]) {
8     pid_t childpid; /* indicates process should spawn another */
9     int fd[2]; /* file descriptors returned by pipe */
10    int nprocs; /* total number of processes in ring */
11
12    if ( (argc != 2) || ((nprocs = atoi(argv[1])) <= 0) ) { /* check command line for a valid number of processes */
13        fprintf(stderr, "Usage: %s nprocs\n", argv[0]);
14        return 1;
15    }
16    if (pipe(fd) == -1) { /* connect std input to std output via a pipe */
17        perror("Failed to create starting pipe");
18        return 1;
19    }
20    if ((dup2(fd[0], STDIN_FILENO) == -1) || (dup2(fd[1], STDOUT_FILENO) == -1)) {
21        perror("Failed to connect pipe");
22        return 1;
23    }
24    if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
25        perror("Failed to close extra descriptors");
26        return 1;
27    }
28    for (i = 1; i < nprocs; i++) { /* create the remaining processes */
29        if (pipe(fd) == -1) {
30            fprintf(stderr, "[%ld]:failed to create pipe %d: %s\n", (long)getpid(), i, strerror(errno));
31            return 1;
32        }
33        if ((childpid = fork()) == -1) {
34            fprintf(stderr, "[%ld]:failed to create child %d: %s\n", (long)getpid(), i, strerror(errno));
35            return 1;
36        }
37        if (childpid > 0) /* for parent process, reassign stdout */
38            error = dup2(fd[1], STDOUT_FILENO);
39        else /* for child process, reassign stdin */
40            error = dup2(fd[0], STDIN_FILENO);
41        if (error == -1) {
42            fprintf(stderr, "[%ld]:failed to dup pipes for iteration %d: %s\n", (long)getpid(), i, strerror(errno));
43            return 1;
44        }
45        if ((close(fd[0]) == -1) || (close(fd[1]) == -1)) {
46            fprintf(stderr, "[%ld]:failed to close extra descriptors %d: %s\n", (long)getpid(), i, strerror(errno));
47            return 1;
48        }
49        if (childpid) break;
50    }
51    fprintf(stderr, "This is process %d with ID %ld and parent id %ld\n", i, (long)getpid(), (long)getppid());
52    return 0;
53 }
54

```

```

1 /* multifork.c (c) 2005-2009 Rahmat M. Samik-Ibrahim, GPL-like */
2 /* ***** */
3 #include <sys/types.h>
4 #include <sys/wait.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <unistd.h>
8 #define DISPLAY1 "PID INDUK** pid (%5.5d) ** *****\n"
9 #define DISPLAY2 "val1(%5.5d) -- val2(%5.5d) -- val3(%5.5d)\n"
10 /* ***** main ** */
11 int main(void) {
12     pid_t val1, val2, val3;
13     printf(DISPLAY1, (int) getpid());
14     fflush(stdout);
15     val1 = fork();
16     waitpid(-1, NULL, 0);
17     val2 = fork();
18     waitpid(-1, NULL, 0);
19     val3 = fork();
20     waitpid(-1, NULL, 0);
21     printf(DISPLAY2, (int) val1, (int) val2, (int) val3);
22     return 0;
23 }
24
25

```

8. Escriba un programa en C, incluya las impresiones a pantalla necesarias para determinar ¿cuál es el árbol de procesos que se genera al escribir como cuerpo del programa la siguiente línea?

```
x = fork() + fork()
```

9.- Complete el siguiente cuadro en un archivo texto con nombre *procesos1.txt*

Programa	Número de Procesos
<i>pfan.c</i>	...
<i>pchain.c</i>	...
<i>pring.c</i>	...
<i>multifork.c</i>	...

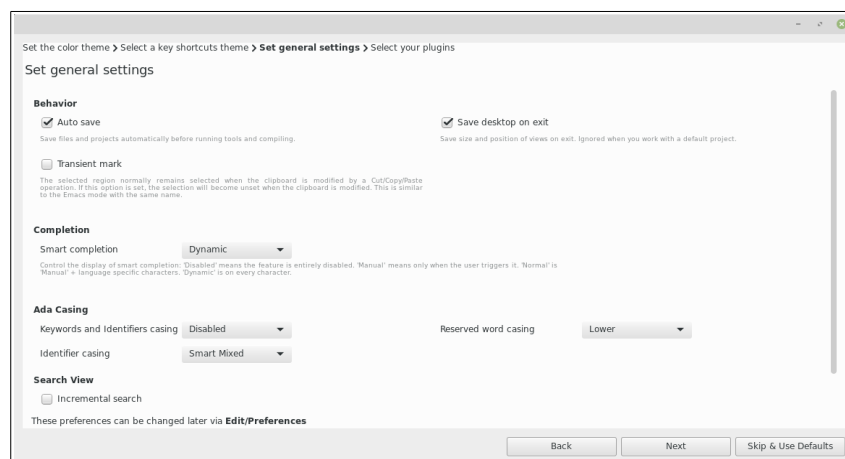
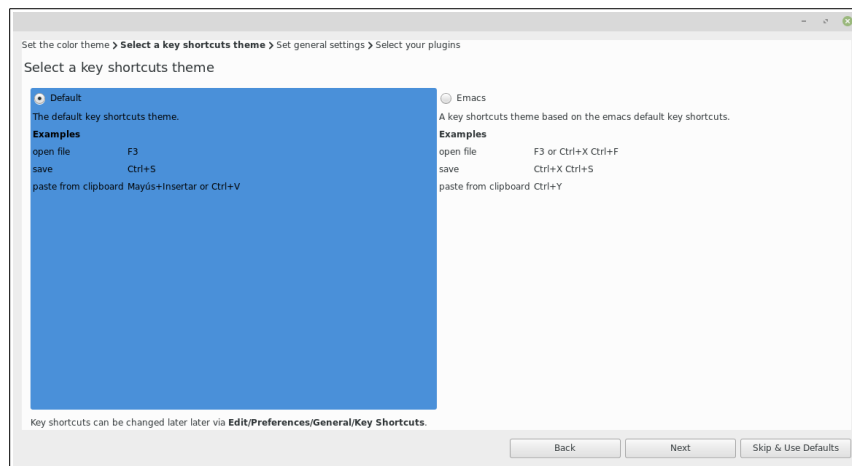
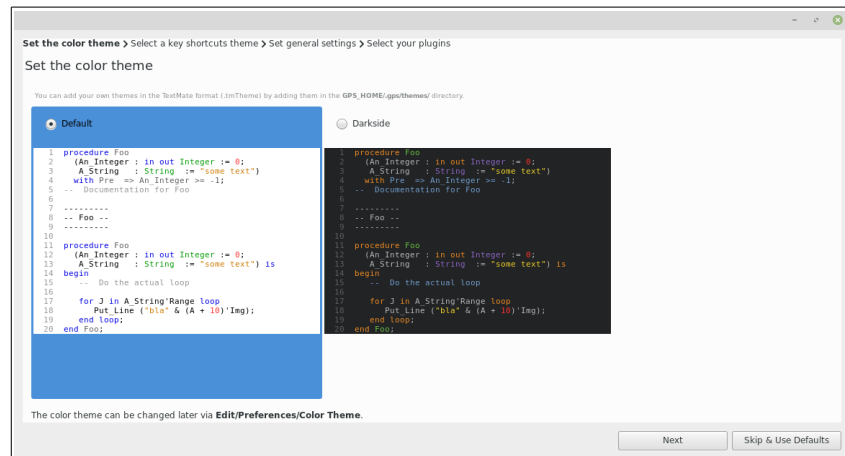
10. Elabore un árbol de procesos, por cada programa, a semejanza de un árbol de directorios. La respuesta debe estar en un archivo texto con nombre *procesos2.txt*

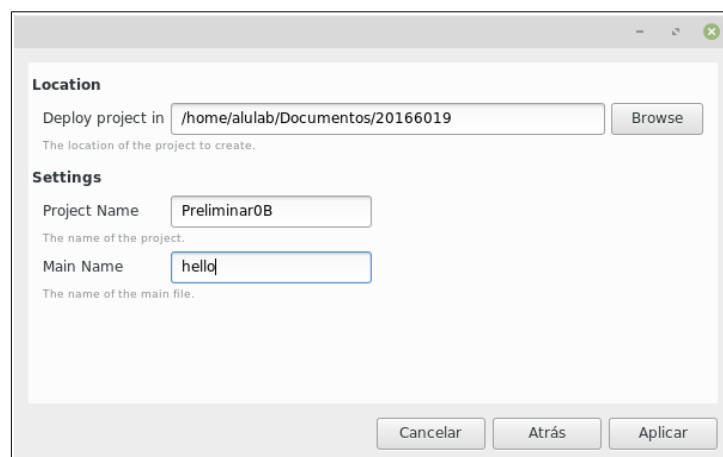
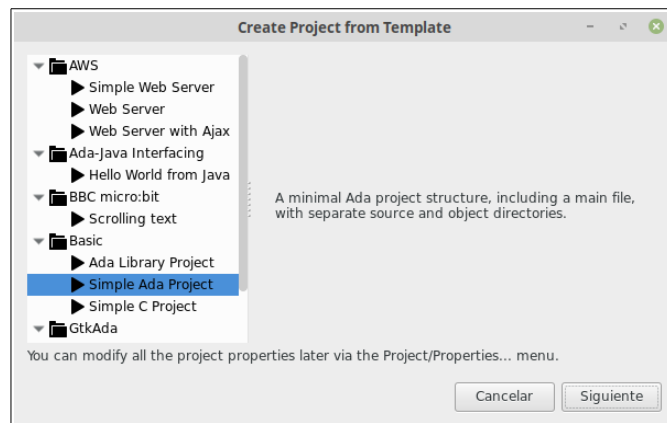
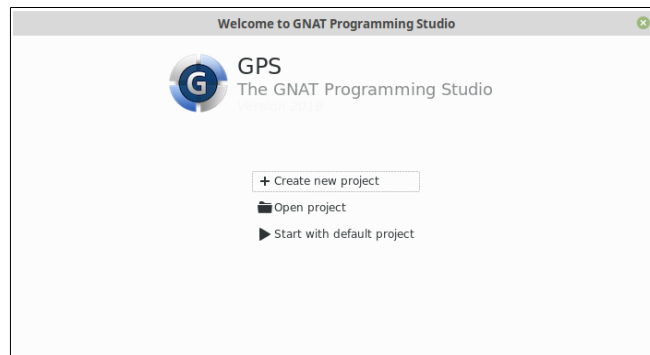
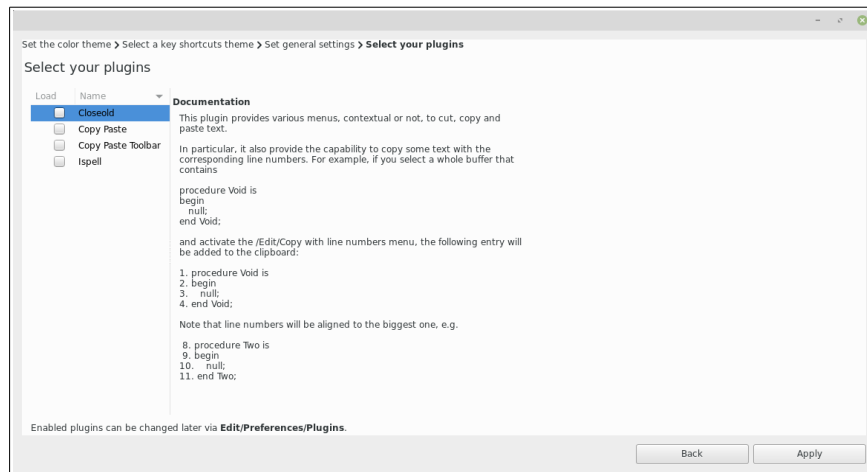
Lenguaje de Programación ADA

Una IDE ampliamente usada es *GNAT Programming Studio* (GPS), nuestros primeros programas los llevaremos a cabo en este entorno.

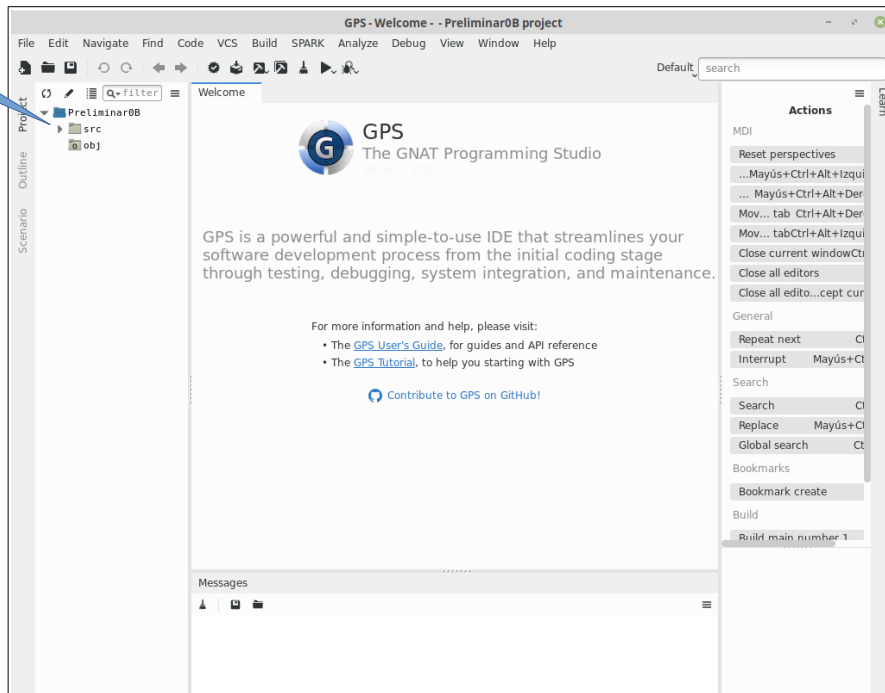
Desde la opción **Programación** del menú de *Linux Mint* elija *Gnat-GPS*

Si es la **primera vez** que es invocado, se mostrará las siguientes ventanas. Elija las opciones que mejor le acomoden:

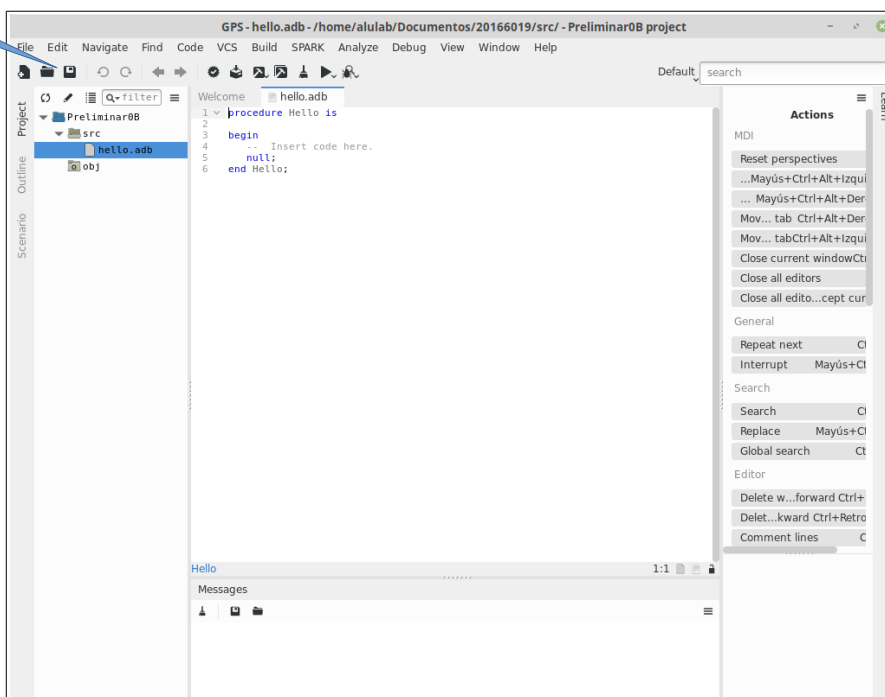




Doble click en src



Grabe el programa



Para compilar el programa, de la barra de herramientas elija: Build Main hello.adb (🔗)



Para ejecutar el programa, de la barra de herramientas elija: Build & Run hello.adb (▶)



A continuación se muestran diferentes estructuras en ADA para que usted pueda escribir programas más complejos.

If else

```
if condition then
    statement;
else
    other statement;
end if;
```

```
if condition then
    statement;
elsif condition then
    other statement;
elsif condition then
    other statement;
...
else
    another statement;
end if;
```

```
with Ada.Text_IO;
use  Ada.Text_IO;
...
type Degrees is new Float range -273.15 .. Float'Last;
...
Temperature : Degrees;
...
if Temperature >= 40.0 then
    Put_Line ("Wow!");
    Put_Line ("It's extremely hot");
elsif Temperature >= 30.0 then
    Put_Line ("It's hot");
elsif Temperature >= 20.0 then
    Put_Line ("It's warm");
elsif Temperature >= 10.0 then
    Put_Line ("It's cool");
elsif Temperature >= 0.0 then
    Put_Line ("It's cold");
else
    Put_Line ("It's freezing");
end if;
```

Case

```
case X is
  when 1 =>
    Walk_The_Dog;
  when 5 =>
    Launch_Nuke;
  when 8 | 10 =>
    Sell_All_Stock;
  when others =>
    Self_Destruct;
end case;
```

Lazos infinitos

```
Endless_Loop :
loop
  Do_Something;
end loop Endless_Loop;
```

Lazos con condición al inicio

```
While_Loop :
while X <= 5 loop
  X := Calculate_Something;
end loop While_Loop;
```

Lazos con condición al final

```
Until_Loop :
loop
  X := Calculate_Something;
  exit Until_Loop when X > 5;
end loop Until_Loop;
```

Lazos con condición en medio

```
Exit_Loop :
loop
  X := Calculate_Something;
  exit Exit_Loop when X > 5;
  Do_Something (X);
end loop Exit_Loop;
```

Lazos for

```
For_Loop :
for I in Integer range 1 .. 10 loop
  Do_Something (I)
end loop For_Loop;
```

Una combinación de formatos

```

if Boolean expression then
  statements
elsif Boolean expression then
  statements
else
  statements
end if;

```

```

while Boolean expression loop
  statements
end loop;

```

```

for variable in range loop
  statements
end loop;

```

```

declare
  declarations
begin
  statements
exception
  handlers
end;

```

```

procedure P (parameters : in out type) is
  declarations
begin
  statements
exception
  handlers
end P;

```

```

function F (parameters : in type) return type is
  declarations
begin
  statements
exception
  handlers
end F;

```

```

package P is
  declarations
private
  declarations
end P;

```

Tipos de datos predefinidos

Predefined types

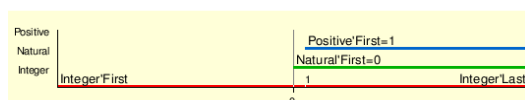
[\[edit\]](#)

There are several predefined types, but most programmers prefer to define their own, application-specific types. Nevertheless, these predefined types are very useful as interfaces between libraries developed independently. The predefined library, obviously, uses these types too.

These types are predefined in the [Standard](#) package:

Integer

This type covers at least the range $-2^{15} + 1 \dots +2^{15} - 1$ (RM 3.5.4 (21) [\(Annotated ⚡\)](#)). The Standard also defines Natural and Positive subtypes of this type.

**Float**

There is only a very weak implementation requirement on this type (RM 3.5.7 (14) [\(Annotated ⚡\)](#)); most of the time you would define your own floating-point types, and specify your precision and range requirements.

Duration

A [fixed point type](#) used for timing. It represents a period of time in seconds (RM A.1 (43) [\(Annotated ⚡\)](#)).

Character

A special form of [Enumerations](#). There are three predefined kinds of character types: 8-bit characters (called Character), 16-bit characters (called Wide_Character), and 32-bit characters (Wide_Wide_Character). Character has been present since the first version of the language ([Ada 83](#)), Wide_Character was added in [Ada 95](#), while the type Wide_Wide_Character is available with [Ada 2005](#).

String

Three indefinite [array types](#), of Character, Wide_Character, and Wide_Wide_Character respectively. The standard library contains packages for handling strings in three variants: fixed length ([Ada.Strings.Fixed](#)), with varying length below a certain upper bound ([Ada.Strings.Bounded](#)), and unbounded length ([Ada.Strings.Unbounded](#)). Each of these packages has a Wide_ and a Wide_Wide_ variant.

Boolean

A Boolean in Ada is an [Enumeration](#) of False and True with special semantics.

Ejercicios

- 1.- Elabore un programa en ADA que reciba un número e imprima un mensaje si el número es primo o no.
- 2.- Elabore un programa en ADA que reciba un número e imprima todos los primos desde 2 hasta dicho número.
- 3.- Elabore un programa en ADA que reciba un número y devuelva el número Fibonacci de dicho número. No emplee la versión recursiva.
- 4.- Elabore un programa en ADA que reciba un número y devuelva el factorial de dicho número. No emplee la versión recursiva.
- 5.- Elabore un programa en ADA que reciba una cadena e imprima la cadena en forma inversa.
- 6.- Elabore un programa en ADA que halle el MCD de dos números, empleando el algoritmo de Euclides.
- 7.- Elabore un programa en ADA que se ingresa un número e imprima su representación en base 2.
- 8.- Elabore un programa en ADA que dado un número imprima el triángulo de PASCAL. El número indica la cantidad de filas.

Compacte la carpeta con nombre igual a su código de alumno, como un archivo con extensión **zip** y coloque el archivo en Intranet en INF239 en la carpeta **Buzón**.

Prof. Alejandro Bello Ruiz