Introducción

Herramientas y compilación

Seminario de Lenguajes opción C

Facultad de Informática
Universidad Nacional de la Plata

2017

Indice

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Modalidad de la cursada

- Consultas los martes, miércoles y viernes.
- Explicación de práctica en los horarios de consulta.
- Asistencia recomendada.
- Aprobación de la cursada:
  - 4 parcialitos de asistencia obligatoria en sala de PC (se rinden en el horario de práctica elegido).
  - Entrega de un trabajo integrador con coloquio en el turno elegido (grupos de 2 personas).
  - 2 fechas de recuperatorio escrito para los temas desaprobados.
- Comisiones para parcialitos y coloquios.

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Plataformas usadas

- Moodle: `https://catedras.info.unlp.edu.ar`
- Todos los anuncios, fechas de entregas y fechas de coloquios se publicarán únicamente a través del Moodle de la cátedra.
- El Moodle tiene un foro para consultas de práctica.
- Twitter de la cátedra: `@seminariocunlp`
- Problemas administrativos (JTPs):
  - Andrés Barbieri: `barbieri@cespi.unlp.edu.ar`
  - Nahuel Cuesta Luengo: `ncuesta@cespi.unlp.edu.ar`
  - Fernando López: `flopez@linti.unlp.edu.ar`

Normas de convivencia

- Las consultas en el foro deben ser claras y breves.
- Evitar subir imágenes grandes al foro.
- Se eliminará del foro cualquier comentario ofensivo o que no esté relacionado con la materia.
- No se aceptarán trabajos prácticos entregados por correo electrónico ni por otros medios.
- Los ayudantes no contestan consultas por e-mail. Usar las consultas presenciales o el foro.
- En el coloquio no se aceptarán versiones del trabajo posteriores a la fecha de entrega.

Estructura mínima de un programa


```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hola\n");
    return 0;
}
```

- Se distinguen mayúsculas y minúsculas.
- La función `main()` tiene el código principal.
- Un programa debe tener exactamente una función main.
- `main()` retorna un entero.
- Por convención `main()` retorna 0 si no hay errores.
- Otras funciones deben declararse antes del main.

Estructura mínima de un programa


```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hola\n");
    return 0;
}
```

- Se distinguen mayúsculas y minúsculas.
- La función `main()` tiene el código principal.
- Un programa debe tener exactamente una función main.
- `main()` retorna un entero.
- Por convención `main()` retorna 0 si no hay errores.
- Otras funciones deben declararse antes del main.

Estructura mínima de un programa


```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hola\n");
    return 0;
}
```

- Se distinguen mayúsculas y minúsculas.
- La función `main()` tiene el código principal.
- Un programa debe tener exactamente una función main.
- `main()` retorna un entero.
- Por convención `main()` retorna 0 si no hay errores.
- Otras funciones deben declararse antes del main.

Estructura mínima de un programa


```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hola\n");
    return 0;
}
```

- Se distinguen mayúsculas y minúsculas.
- La función `main()` tiene el código principal.
- Un programa debe tener exactamente una función main.
- `main()` retorna un entero.
- Por convención `main()` retorna 0 si no hay errores.
- Otras funciones deben declararse antes del main.

Estructura mínima de un programa


```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hola\n");
    return 0;
}
```

- Se distinguen mayúsculas y minúsculas.
- La función `main()` tiene el código principal.
- Un programa debe tener exactamente una función main.
- `main()` retorna un entero.
- Por convención `main()` retorna 0 si no hay errores.
- Otras funciones deben declararse antes del main.

Estructura mínima de un programa


```c
#include <stdio.h>

int main(int argc, char *argv[]){
    printf("Hola\n");
    return 0;
}
```

- Se distinguen mayúsculas y minúsculas.
- La función `main()` tiene el código principal.
- Un programa debe tener exactamente una función main.
- `main()` retorna un entero.
- Por convención `main()` retorna 0 si no hay errores.
- Otras funciones deben declararse antes del main.

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Tipos de datos simples

- `int` → enteros.
- `short` → enteros (puede tener menos bytes que un int)
- `long` → enteros (puede tener más bytes que un int)
- `long long` → enteros (puede tener más bytes que un long)
- `char` → caracteres, 1 byte cada uno (K&R).
- `float` → punto flotante.
- `double` → punto flotante doble presición.
- `long double` → punto flotante cuadruple presición.
- No hay booleanos: cualquier entero **distinto** a 0 es TRUE.
- Punteros (los vamos a ver más adelante).

Característica interesantes del lenguaje

Pre y pos incremento/decremento


```
int a, b, c, d;
a = 5;
b = 2;
b--;
c = ++a;
d = b++;
```

- $a \leftarrow 5$
- $b \leftarrow 2$
- $b \leftarrow b - 1$
- $a \leftarrow a + 1; \quad c \leftarrow a$
- $d \leftarrow b; \quad b \leftarrow b + 1$

Característica interesantes del lenguaje

Pre y pos incremento/decremento


```
int a, b, c, d;
a = 5;
b = 2;
b--;
c = ++a;
d = b++;
```

- $a \leftarrow 5$
- $b \leftarrow 2$
- $b \leftarrow b - 1$
- $a \leftarrow a + 1; \quad c \leftarrow a$
- $d \leftarrow b; \quad b \leftarrow b + 1$

Característica interesantes del lenguaje

Pre y pos incremento/decremento


```
int a, b, c, d;
a = 5;
b = 2;
b--;
c = ++a;
d = b++;
```

- $a \leftarrow 5$
- $b \leftarrow 2$
- $b \leftarrow b - 1$
- $a \leftarrow a + 1; \quad c \leftarrow a$
- $d \leftarrow b; \quad b \leftarrow b + 1$

Característica interesantes del lenguaje

Pre y pos incremento/decremento


```
int a, b, c, d;
a = 5;
b = 2;
b--;
c = ++a;
d = b++;
```

- $a \leftarrow 5$
- $b \leftarrow 2$
- $b \leftarrow b - 1$
- $a \leftarrow a + 1; \quad c \leftarrow a$
- $d \leftarrow b; \quad b \leftarrow b + 1$

Característica interesantes del lenguaje

Pre y pos incremento/decremento


```
int a, b, c, d;
a = 5;
b = 2;
b--;
c = ++a;
d = b++;
```

- $a \leftarrow 5$
- $b \leftarrow 2$
- $b \leftarrow b - 1$
- $a \leftarrow a + 1; \quad c \leftarrow a$
- $d \leftarrow b; \quad b \leftarrow b + 1$

Característica interesantes del lenguaje

Pre y pos incremento/decremento


```
int a, b, c, d;
a = 5;
b = 2;
b--;
c = ++a;
d = b++;
```

- $a \leftarrow 5$
- $b \leftarrow 2$
- $b \leftarrow b - 1$
- $a \leftarrow a + 1; \quad c \leftarrow a$
- $d \leftarrow b; \quad b \leftarrow b + 1$

Característica interesantes del lenguaje

Tipado débil (conversiones automáticas)


```c
int a; // Asumamos que sizeof(int) == 4
short int b; // Asumamos que sizeof(short int) == 2
a = 100000;
b = a; // ¿Cuanto vale b?
```

- $a \leftarrow 00000000000000011000011010100000_2$
- $b \leftarrow 1000011010100000_2$
- Eso es 34464_{10}, pero...
- Son enteros con signo en complemento 2
- Por lo tanto $b \leftarrow -31072_{10}$

Característica interesantes del lenguaje

Tipado débil (conversiones automáticas)


```c
int a; // Asumamos que sizeof(int) == 4
short int b; // Asumamos que sizeof(short int) == 2
a = 100000;
b = a; // ¿Cuanto vale b?
```

- $a \leftarrow 00000000000000011000011010100000_2$
- $b \leftarrow 1000011010100000_2$
- Eso es 34464_{10}, pero...
- Son enteros con signo en complemento 2
- Por lo tanto $b \leftarrow -31072_{10}$

Característica interesantes del lenguaje

Tipado débil (conversiones automáticas)


```c
int a; // Asumamos que sizeof(int) == 4
short int b; // Asumamos que sizeof(short int) == 2
a = 100000;
b = a; // ¿Cuanto vale b?
```

- $a \leftarrow 00000000000000011000011010100000_2$
- $b \leftarrow 1000011010100000_2$
- Eso es 34464_{10}, pero...
- Son enteros con signo en complemento 2
- Por lo tanto $b \leftarrow -31072_{10}$

Característica interesantes del lenguaje

Tipado débil (conversiones automáticas)


```c
int a; // Asumamos que sizeof(int) == 4
short int b; // Asumamos que sizeof(short int) == 2
a = 100000;
b = a; // ¿Cuanto vale b?
```

- $a \leftarrow 00000000000000011000011010100000_2$
- $b \leftarrow 1000011010100000_2$
- Eso es 34464_{10}, pero...
- Son enteros con signo en complemento 2
- Por lo tanto $b \leftarrow -31072_{10}$

Característica interesantes del lenguaje

Tipado débil (conversiones automáticas)


```c
int a; // Asumamos que sizeof(int) == 4
short int b; // Asumamos que sizeof(short int) == 2
a = 100000;
b = a; // ¿Cuanto vale b?
```

- $a \leftarrow 00000000000000011000011010100000_2$
- $b \leftarrow 1000011010100000_2$
- Eso es 34464_{10}, pero...
- Son enteros con signo en complemento 2
- Por lo tanto $b \leftarrow -31072_{10}$

Característica interesantes del lenguaje

Tipado débil (conversiones automáticas)

```
int a; // Asumamos que sizeof(int) == 4
short int b; // Asumamos que sizeof(short int) == 2
a = 100000;
b = a; // ¿Cuanto vale b?
```

- $a \leftarrow 00000000000000011000011010100000_2$
- $b \leftarrow 1000011010100000_2$
- Eso es 34464_{10}, pero...
- Son enteros con signo en complemento 2
- Por lo tanto $b \leftarrow -31072_{10}$

Característica interesantes del lenguaje

Conversiones explícitas (castings)


```
float a = 3.5;
float b;
int c;
char d;
b = (int) a;
c = a;
d = a;
```

- $a \leftarrow 3.5$
- $b \leftarrow 3$
- $c \leftarrow 3$
- $d \leftarrow 3$

Característica interesantes del lenguaje

Conversiones explícitas (castings)


```
float a = 3.5;
float b;
int c;
char d;
b = (int) a;
c = a;
d = a;
```

- $a \leftarrow 3.5$
- $b \leftarrow 3$
- $c \leftarrow 3$
- $d \leftarrow 3$

Característica interesantes del lenguaje

Conversiones explícitas (castings)


```
float a = 3.5;
float b;
int c;
char d;
b = (int) a;
c = a;
d = a;
```

- $a \leftarrow 3.5$
- $b \leftarrow 3$
- $c \leftarrow 3$
- $d \leftarrow 3$

Característica interesantes del lenguaje

Conversiones explícitas (castings)


```
float a = 3.5;
float b;
int c;
char d;
b = (int) a;
c = a;
d = a;
```

- $a \leftarrow 3.5$
- $b \leftarrow 3$
- $c \leftarrow 3$
- $d \leftarrow 3$

Característica interesantes del lenguaje

Conversiones explícitas (castings)


```
float a = 3.5;
float b;
int c;
char d;
b = (int) a;
c = a;
d = a;
```

- $a \leftarrow 3.5$
- $b \leftarrow 3$
- $c \leftarrow 3$
- $d \leftarrow 3$

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

char es un tipo entero


```
char x = 'a';
x++;
x = x - 32;
```

- $x \leftarrow 97_{10}$ (ASCII de 'a')
- $x \leftarrow 98_{10}$ (ASCII de 'b')
- $x \leftarrow 66_{10}$ (ASCII de 'B')

¿Es minúscula?

```
if (x >= 'a' && x <= 'z') { //...
```

Investigar las funciones de la familia `islower()` con el comando:

```
$ man islower
```

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: &&, ||, !.

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: `&&`, `||`, `!`.

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: `&&`, `||`, `!`.

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: `&&`, `||`, `!`.

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: `&&`, `||`, `!`.

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: `&&`, `||`, `!`.

Característica interesantes del lenguaje

Operadores bit a bit


```c
unsigned int a; // Suponemos sizeof(unsigned) == 4
a = ~0;       // not
a = 1 & 5     // and
a = 1 | 4     // or
a = 1 ^ 3     // xor
a = 1 << 3    // shift a izquierda
a = 8 >> 1    // shift a derecha
```

- $\sim 0 = 0xFFFFFFFF$
- $001_2 \wedge 101_2 = 001_2$
- $001_2 \vee 100_2 = 101_2$
- $001_2 \oplus 011_2 = 010_2$
- $0001_2 \ll 3 = 1000_2$
- $1000_2 \gg 1 = 0100_2$

No confundir con los operadores lógicos: `&&`, `||`, `!`.

Estructuras de control elementales


```
if (condición) {
    // si condición != 0
}
else {
    // si condición == 0
}

while (condición) { } // Mientras condición != 0

for (inicialización; condición; incremento) { }
```

Ver ejemplos.

Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

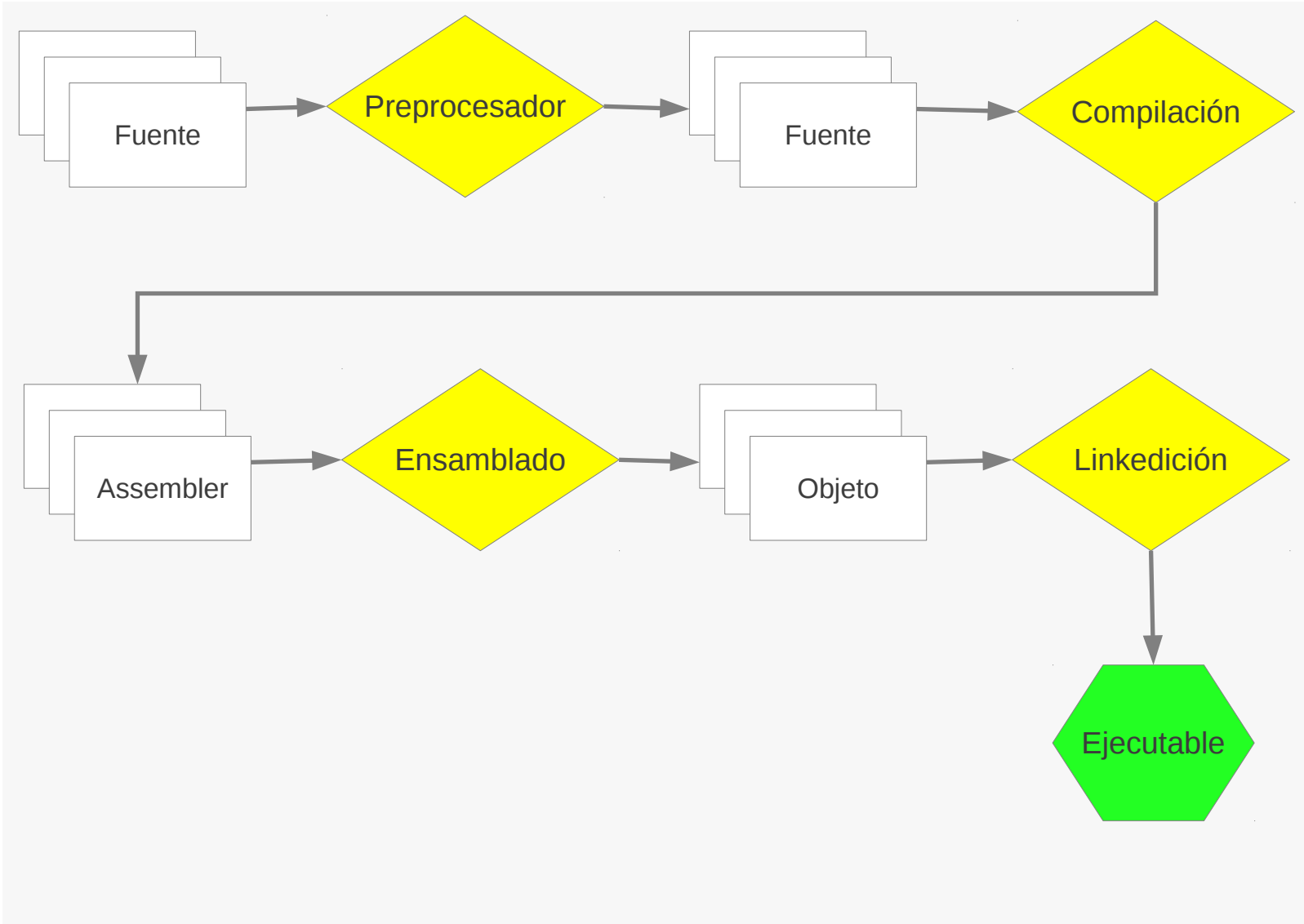
Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

Operadores lógicos y comparaciones

- && AND lógico.
- || OR lógico.
- ! NOT lógico.
- == igualdad.
- != desigualdad.
- < y > menor y mayor.
- <= y >= menor-o-igual y mayor-o-igual.

Proceso de compilación



Preprocesador

- Transforma el código.
- Substituye texto.
- Es independiente del lenguaje C pero indispensable.
- Procesa las directivas que empiezan con #.
- Ejemplos:

```
#include <stdio.h>
#include "ejemplo.h"
#define test 5
```

Preprocesador

- Transforma el código.
- Substituye texto.
- Es independiente del lenguaje C pero indispensable.
- Procesa las directivas que empiezan con #.
- Ejemplos:

```
#include <stdio.h>
#include "ejemplo.h"
#define test 5
```

Preprocesador

- Transforma el código.
- Substituye texto.
- Es independiente del lenguaje C pero indispensable.
- Procesa las directivas que empiezan con #.
- Ejemplos:

```
#include <stdio.h>
#include "ejemplo.h"
#define test 5
```

Preprocesador

- Transforma el código.
- Substituye texto.
- Es independiente del lenguaje C pero indispensable.
- Procesa las directivas que empiezan con #.
- Ejemplos:

```
#include <stdio.h>
#include "ejemplo.h"
#define test 5
```

Preprocesador

- Transforma el código.
- Substituye texto.
- Es independiente del lenguaje C pero indispensable.
- Procesa las directivas que empiezan con #.
- Ejemplos:

```
#include <stdio.h>
#include "ejemplo.h"
#define test 5
```

Compilador

- Verifica la sintaxis del programa.
- Advierte sobre potenciales errores semánticos.
- Transforma el código C en assembler.
- Puede optimizar el código generado.
- Puede agregar información extra para el debugger.

Compilador

- Verifica la sintaxis del programa.
- Advierte sobre potenciales errores semánticos.
- Transforma el código C en assembler.
- Puede optimizar el código generado.
- Puede agregar información extra para el debugger.

Compilador

- Verifica la sintaxis del programa.
- Advierte sobre potenciales errores semánticos.
- Transforma el código C en assembler.
- Puede optimizar el código generado.
- Puede agregar información extra para el debugger.

Compilador

- Verifica la sintaxis del programa.
- Advierte sobre potenciales errores semánticos.
- Transforma el código C en assembler.
- Puede optimizar el código generado.
- Puede agregar información extra para el debugger.

Compilador

- Verifica la sintaxis del programa.
- Advierte sobre potenciales errores semánticos.
- Transforma el código C en assembler.
- Puede optimizar el código generado.
- Puede agregar información extra para el debugger.

Ensamblador y link-editor

Ensamblador:

- Traduce código assembler a código de máquina.
- El binario generado se denomina "código objeto".

Link-editor

- Genera un archivo ejecutable a partir de uno o más códigos objeto.
- Se puede dividir el programa en módulos y compilar cada uno por separado, al modificar un módulo sólo hay que compilar ese módulo y volver a "linkearlo" con los otros códigos objeto.
- El proceso anterior ahorra tiempo en proyectos grandes.

Ensamblador y link-editor

Ensamblador:

- Traduce código assembler a código de máquina.
- El binario generado se denomina “código objeto”.

Link-editor

- Genera un archivo ejecutable a partir de uno o más códigos objeto.
- Se puede dividir el programa en módulos y compilar cada uno por separado, al modificar un módulo sólo hay que compilar ese módulo y volver a “linkearlo” con los otros códigos objeto.
- El proceso anterior ahorra tiempo en proyectos grandes.

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.

Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

GCC

El "GNU C Compiler", además de compilar, puede invocar a las otras herramientas. Invocando a GCC se puede hacer una sola etapa de las anteriores, o todas las etapas de una vez.
Argumentos de gcc:

- `-E` → Preprocesa.
- `-S` → Compila (genera código assembler)
- `-c` → Ensambla (genera código objeto).
- `-o` → Permite indicar el nombre del archivo de salida.
- Sin argumentos realiza todos los pasos.
- `-std=c99` → Estándar C usado.
- `-Wall` → Muestra las advertencias.

Ejemplos

Entornos de desarrollo integrados (IDEs) y editores

Existen IDEs para todos los gustos:

- Anjuta
- Codeblocks
- KDevelop
- NetBeans
- Eclipse

Pero también hay editores para programadores:

- Gedit (recomendado)
- Geany
- Vim
- JEdit

La cátedra no fuerza el uso de ninguna de estas opciones, pero es obligatorio saber compilar con gcc desde la terminal.

Entornos de desarrollo integrados (IDEs) y editores

Existen IDEs para todos los gustos:

- Anjuta
- Codeblocks
- KDevelop
- NetBeans
- Eclipse

Pero también hay editores para programadores:

- Gedit (recomendado)
- Geany
- Vim
- JEdit

La cátedra no fuerza el uso de ninguna de estas opciones, pero es obligatorio saber compilar con gcc desde la terminal.

Entornos de desarrollo integrados (IDEs) y editores

Existen IDEs para todos los gustos:

- Anjuta
- Codeblocks
- KDevelop
- NetBeans
- Eclipse

Pero también hay editores para programadores:

- Gedit (recomendado)
- Geany
- Vim
- JEdit

La cátedra no fuerza el uso de ninguna de estas opciones, pero es obligatorio saber compilar con gcc desde la terminal.

Terminal/Consola

Comandos básicos (dentro de Cygwin es lo mismo):

- `pwd` → Directorio actual.
- `ls` → Muestra el contenido de un directorio.
- `cat archivo` → Muestra el contenido del archivo.
- `cd directorio` → Se posiciona en el directorio indicado.
- `cd ..` → Vuelve al directorio padre.
- `man 3 printf`
- `man ls` o `man gcc`
- `help cd`
- `programa_instalado`
- `./programa_sin_instalar`
- manpages o `http://man.cx`

Resumen

- Cómo se aprueba la cursada.
- Qué hace el preprocesador/compilador/ensamblador/linker.
- Cómo invocarlos desde la terminal.
- ¿Es necesario aprender a usar algún IDE?.
- ¿Es necesario que aprenda a invocar gcc desde la terminal?
- ¿Cómo se escribe un programa básico?