Convenciones para el código

1. Introducción

Al trabajar con un lenguaje de programación, tenemos cierta flexibilidad para escribir nuestros programas. Generalmente, se adopta ciertas convenciones respecto a diferentes aspectos como la organización de archivos, la indentación, los comentarios, las declaraciones, los espacios en blanco, cómo nombrar variables y funciones, etc, de manera de seguir siempre un mismo criterio, mejorando la legibilidad y mantenimiento del código fuente.

2. Propuesta

Desde de la cátedra, proponemos el siguiente criterio:

a) **Indentación:** 2 espacios por nivel de indentación. Nunca usar tabulaciones. Por ejemplo:

```
int main() {
  int i, j;
  for (i = 0; i < 10; ++i) {
    j = i;
    while (j > 0)
      j--;
    if (j != 0)
      break;
    else
      j = 4;
  }
  return 0;
}
```

b) **Llaves:** Abrimos la llave en la misma línea y la cerramos en una nueva línea. Siempre que podamos, omitimos las llaves. Por ejemplo:

```
int some_fun() {
  ...
  return 0;
}

int main() {

  if (a > 0) {
    ...
  } else {
    ...
  }

  for (i = 0; i < 10; ++i) {
    ...
    while (j > 0) {
      if (k > 0)
```

```
        c++;
      else
        c--;
    }
    for (h = 0; h < 10; ++h)
      c = h + j;
  }
}
```

c) **Espaciado:**

- Siempre dejar un espacio después de un controlador de flujo (como ser: `if, do, while, for, else`, etc) y entre la condición y la llave que se abre. Por ejemplo:

```
for (...) {
  ...
}

if (...) {
  ...
}

while (...) {
  ...
}
```

- Siempre dejar un espacio entre un operador y los operandos. Por ejemplo:

```
if (a > 0 && b != 0) {
  ...
}

res = (k + 5 - z) / 7;
```

(Excepto para el operador `->` . Por ejemplo: `a->campo`)

- Siempre dejar un espacio después de una coma y no antes. Por ejemplo:

```
int fun(char caracter, int cant, int* res) {
  ...
}

double var, count, res = 5.3;
```

d) **Longitud de línea:** máximo de 80 caracteres por línea.

e) **Nomenclatura:**

- **Nombres de funciones:** Utilizaremos identificadores en minúscula separados por un guion bajo. Por ejemplo:

```
void slist_append(...)
double obtener_raiz_cuadrada(...)
```

- **Nombres de variables:** Utilizaremos identificadores en minúscula y cada palabra intermedia comenzará en mayúscula (*camelCase*). Por ejemplo:

```
int testCase;
char* listaDeUsuarios[];
int contador;
```

- **Nombres de constantes:** Utilizaremos identificadores en mayúscula separados por un guion bajo. Por ejemplo:

  ```
  #define MAX_ELEM 10
  ```

- **Nombres de nuevos tipos de datos y alias:** Cada palabra comenzará en mayúscula (*CamelCase*). Por ejemplo:

  ```
  enum TipoDeRecorrido {
    ...
  };

  struct EmpleadoLocal {
    ...
  };

  typedef void (*ImprimirCallback)(int);
  ```

Este criterio no es estricto, el alumno puede tomar un criterio distinto, siempre y cuando **sea consistente en todo el código desarrollado**.

3. GNU Indent

Podemos indentar nuestro código automáticamente utilizando la herramienta GNU: *indent* .
(En debian/ubuntu se puede instalar con: `sudo apt-get install indent`).
Por ejemplo, para actualizar el formato del archivo fuente *somefile.c*, ejecutamos:

```
indent -kr -brf -i2 -l80 -nut somefile.c
```

Las opciones representan:

- **-kr**: estilo *Kernighan & Ritchie*.
- **-brf**: para las funciones, abrir la llave en la misma linea.
- **-i2**: utilizar 2 espacios para cada nivel de indentación.
- **-l80**: restringir las líneas a un máximo de 80 caracteres.
- **-nut**: eliminar las tabulaciones.

Para más información en las opciones disponibles, recurrir al manual (`man indent`).