Clase 1. Continuación/repetición minicurso, variables, lectura, escritura, operadores, compilación, sintaxis y errores.

Calificación

Proporción de calificación

- 40 % tareas.
- 30 % 2 proyectos 10 % el primero y 20% el segundo.
- 30 % Examenes (2 examenes de 15 % cada uno).

Tareas

- Las tareas valen 1 entregadas a tiempo, y 0.2 menos por cada día natural de retraso, a los 5 días ya no cuentan.
  - 0.8 programa (si funciona).
  - 0.2 reporte pequeño:
    - Qué se implemento.
    - Para que sirve.
    - Ejemplo de ejecución de un caso prueba.
    - Conclusiones u observaciones interesantes (en que casos no funciona, porqué, error/comportamiento interesante, forma de eficientarlo,etc.).
- Cada programa debe de estar en una carpeta diferente (puede ser la carpeta del proyecto de codeblocks).
- Cada tarea ira acompañada de un caso de prueba.
- En todos los casos (de la parte de C), se deben de realizar funciones para realizar lo que se pide (es decir, no poner todo en el main).
- Las funciones en todos los casos, deben de ir declaradas en un .h y definidas en un .c.
- Todas las carpetas de los programas de un semana se pondrán en una sola carpeta tareaNN_apellido_XX Donde NN, es el n'úmero consecutivo de la tarea y XX son la(s) iniciales del nombre, ejemplo para Jose Juan Perez, tarea 1, la carpeta se llamaría: tarea01_perez_JJ
- La carpeta comprimida con extension .zip, .tar.gz,.tar.bz o 7z, se envia al ayudante, con el nombre de la carpeta como subject.

Estructura y sintaxis

Estructura del programa en C


```
1 #include <stdio.h>
  int main()
3 {
  int a, b;
5  printf("Llamada a
      funcion");
  return 0;
7 }
```

Un programa en C usualmente tiene :

- Directivas de precompilador, generalmente inclusión de librerias.
- Una y solo una función main.
- Definición de datos (o declaración). La declaración consite en decirle al compilador que existen los datos o funciones, la definición especifica lo que hace la función o requiere la memoria para las variables.
- Una o mas llamadas a función.
- En C hay distinción entre mayusculas y minusculas.

Sintaxis básica


```c
#include <stdio.h>
int main()
{
int a, b;
 printf("Llamada a funcion");
return 0;
}
```

- Las directivas de precompilador comienzan con # y son leidas y transformadas en diferentes valores (variables, constantes, funciones, macros, etc.) antes de compilar, generalmente en la misma llamada al compilador.

Sintaxis básica


```c
#include <stdio.h>
int main()
{
int a, b;
 printf("Llamada a funcion");
return 0;
}
```

- Las directivas de precompilador comienzan con # y son leidas y transformadas en diferentes valores (variables, constantes, funciones, macros, etc.) antes de compilar, generalmente en la misma llamada al compilador.
- Los espacios en blanco o saltos de líneas son ignorados.

Sintaxis básica


```c
#include <stdio.h>
int main()
{
int a, b;
 printf("Llamada a funcion");
return 0;
}
```

- Las directivas de precompilador comienzan con # y son leidas y transformadas en diferentes valores (variables, constantes, funciones, macros, etc.) antes de compilar, generalmente en la misma llamada al compilador.
- Los espacios en blanco o saltos de líneas son ignorados.
- Las llaves { } se utilizan para indicar que un bloque de código pertenecen a la misma rama de ejecución. Por ejemplo: a la misma función, al mismo resultado de un condicional, al mismo ciclo,etc.

Sintaxis básica


```
1 #include <stdio.h>
  int main()
3 {
  int a, b;
5  printf("Llamada a funcion");
  return 0;
7 }
```

- Las directivas de precompilador comienzan con # y son leidas y transformadas en diferentes valores (variables, constantes, funciones, macros, etc.) antes de compilar, generalmente en la misma llamada al compilador.
- Los espacios en blanco o saltos de líneas son ignorados.
- Las llaves { } se utilizan para indicar que un bloque de código pertenecen a la misma rama de ejecución. Por ejemplo: a la misma función, al mismo resultado de un condicional, al mismo ciclo,etc.
- Los enunciados (instrucciones) se separan con un punto y coma.

Sintaxis básica


```c
#include <stdio.h>
int main()
{
int a, b;
 printf("Llamada a funcion");
return 0;
}
```

- Las directivas de precompilador comienzan con # y son leidas y transformadas en diferentes valores (variables, constantes, funciones, macros, etc.) antes de compilar, generalmente en la misma llamada al compilador.
- Los espacios en blanco o saltos de líneas son ignorados.
- Las llaves { } se utilizan para indicar que un bloque de código pertenecen a la misma rama de ejecución. Por ejemplo: a la misma función, al mismo resultado de un condicional, al mismo ciclo,etc.
- Los enunciados (instrucciones) se separan con un punto y coma.
- los comentarios llevan una inician y terminan con un diagonal y asterisco: /*Este es un comentario */.

Sintaxis básica


```c
#include <stdio.h>
int main()
{
int a, b;
 printf("Llamada a funcion");
return 0;
}
```

- Las directivas de precompilador comienzan con # y son leidas y transformadas en diferentes valores (variables, constantes, funciones, macros, etc.) antes de compilar, generalmente en la misma llamada al compilador.
- Los espacios en blanco o saltos de líneas son ignorados.
- Las llaves { } se utilizan para indicar que un bloque de código pertenecen a la misma rama de ejecución. Por ejemplo: a la misma función, al mismo resultado de un condicional, al mismo ciclo,etc.
- Los enunciados (instrucciones) se separan con un punto y coma.
- los comentarios llevan una inician y terminan con un diagonal y asterisco: /*Este es un comentario */.
- Las palabras reservadas no pueden ser utilizadas para identificar funciones o variables.

Archivos de cabecera

La instrucción:

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

- Una directiva de precompilador (o preprocesador), comienza con el símbolo #.
- Indica que se deben de hacer modificaciones al programa antes de compilarlo. Un programa anterior al compilador (pero generalmente llamado por el compilador), lee estas directivas y lo modifica.
- En este caso indica que se deben incluir el archivo stdio.h en este programa.
- Esta libreria *estándar*, contiene declaraciones de funciones, las definiciones son incluidas por el compilador en tiempo de compilación, se encuentran generalmente en precompilados .o (en linux u .obj en windows) y cuando son definiciones de librerias estándar se instalan con el compilador y no es necesario incluirlas explícitamente.

Palabras reservadas

Las palabras reservadas son un conjunto de palabras que no pueden ser utilizadas como identificadores (nombres de funciones o variables), y ya tienen una utilidad definida en el lenguaje. Ejemplos:

auto, if, break, int, case, long, char, register, continue, return, default, short, do, sizeof, double, static, else, struct, entry, switch, extern, typedef, float, union, for, unsigned, goto, enum, void, const, signed, volatile.

Tipos de datos

Tipos de datos

En C existen diferentes tipos de datos:

- Tipos de datos básicos. Son los tipos enteros (de diferente tipo y tamaño, el char en realida es entero), y flotantes (de diferente precisión).

Tipos de datos

En C existen diferentes tipos de datos:

- Tipos de datos básicos. Son los tipos enteros (de diferente tipo y tamaño, el char en realida es entero), y flotantes (de diferente precisión).
- Tipos de datos enumerados. Son tipos de variables que tienen valores específicos, siempre son definidos por el usuario y C los transforma a enteros, comenzando en 0, están pensados como etiquetas o datos categoricos, por lo que no se (deben) utilizan para operaciones numéricas.

Tipos de datos

En C existen diferentes tipos de datos:

- Tipos de datos básicos. Son los tipos enteros (de diferente tipo y tamaño, el char en realida es entero), y flotantes (de diferente precisión).
- Tipos de datos enumerados. Son tipos de variables que tienen valores específicos, siempre son definidos por el usuario y C los transforma a enteros, comenzando en 0, están pensados como etiquetas o datos categoricos, por lo que no se (deben) utilizan para operaciones numéricas.
- Tipo **void**. Es un tipo de dato que indica que no se tiene un valor disponible (como ausencia de dato). Se utiliza mas comunmente para definir funciones que no tienen un valor de retorno.

Tipos de datos

En C existen diferentes tipos de datos:

- Tipos de datos básicos. Son los tipos enteros (de diferente tipo y tamaño, el char en realida es entero), y flotantes (de diferente precisión).
- Tipos de datos enumerados. Son tipos de variables que tienen valores específicos, siempre son definidos por el usuario y C los transforma a enteros, comenzando en 0, están pensados como etiquetas o datos categoricos, por lo que no se (deben) utilizan para operaciones numéricas.
- Tipo **void**. Es un tipo de dato que indica que no se tiene un valor disponible (como ausencia de dato). Se utiliza mas comunmente para definir funciones que no tienen un valor de retorno.
- Tipos de datos derivados. Apuntadores, arreglos, estructuras y uniones.

Tipos de datos básicos, recordatorio

Los tipos de datos básicos son:

- *entero (32 bits):* **int**,
- *reales de precisión simple(32 bits):* **float**,
- *reales precisión doble(64 bits):* **double**,
- *Carácteres:* **char**,
- *enteros largos (64 bits):* **long int**.

Modificador para enteros sin signo **unsigned**. Modificador para enteros largos **long** y **long long**.

Variables

Variables

- Una variable es el nómbre que se le da a una región para almacenamiento de datos y que puede ser manipulada por el programa.

Variables

- Una variable es el nómbre que se le da a una región para almacenamiento de datos y que puede ser manipulada por el programa.
- Cada variable en C tiene un tipo *específico*.

Variables

- Una variable es el nómbre que se le da a una región para almacenamiento de datos y que puede ser manipulada por el programa.
- Cada variable en C tiene un tipo *específico*.
- Con el tipo se especifica el número de bits para el almacenamiento, el tipo de operaciones que se puede realizar, y el rango de valores que puede tomar las variables. [1]

La forma más común de declarar variables es declararlas y definirlas en la misma instrucción. **Un nombre de variable o función solo puede ser definida 1 solo vez.** Es decir, al momento que se declara se solicita memoria para la misma (es posible declararlas sin definirlas se verá mas adelante en el curso).

[1] Para rango de las variables buscar por limits.h y float.h.

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

- En el ejemplo anterior la variable se definió cuando se declaró. Es decir se le asignó una locación de memoria.

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

- En el ejemplo anterior la variable se definió cuando se declaró. Es decir se le asignó una locación de memoria.
- La variable no está inicializada, tiene un valor no definido.

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

- En el ejemplo anterior la variable se definió cuando se declaró. Es decir se le asignó una locación de memoria.
- La variable no está inicializada, tiene un valor no definido.
- Se pueden inicializar en la declaración o después, pero siempre es deseable inicializar, ej.:
  int numint1=-5, numint2=2, numint3=6;

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

- En el ejemplo anterior la variable se definió cuando se declaró. Es decir se le asignó una locación de memoria.
- La variable no está inicializada, tiene un valor no definido.
- Se pueden inicializar en la declaración o después, pero siempre es deseable inicializar, ej.:
  int numint1=-5, numint2=2, numint3=6;
- El operador de asignación es el **=**.

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

- En el ejemplo anterior la variable se definió cuando se declaró. Es decir se le asignó una locación de memoria.
- La variable no está inicializada, tiene un valor no definido.
- Se pueden inicializar en la declaración o después, pero siempre es deseable inicializar, ej.:
  int numint1=-5, numint2=2, numint3=6;
- El operador de asignación es el **=**.
- Es deseable que las variables tomen un nombre que indique para que se usa. El nombre no debe iniciar con número o caracter especial, y no puede ser una palabra reservada.

Declaración de variables

La forma genérica de declarar variables es:

tipo variable1, variable2, variable3;

Ejemplo: **int** numint1, numint2, numint3;
float x, y, z;

- En el ejemplo anterior la variable se definió cuando se declaró. Es decir se le asignó una locación de memoria.
- La variable no está inicializada, tiene un valor no definido.
- Se pueden inicializar en la declaración o después, pero siempre es deseable inicializar, ej.:
  int numint1=-5, numint2=2, numint3=6;
- El operador de asignación es el **=**.
- Es deseable que las variables tomen un nombre que indique para que se usa. El nombre no debe iniciar con número o caracter especial, y no puede ser una palabra reservada.
- Se produce un error de overflow (integer overflow, floating point overflow) en tiempo de ejecución cuando a una variable se le asigna un valor fuera de su rango, en general, el valor que toma está indefinido.

Tipos de expresiones

Existen 2 tipos de expresiones en C.

- LVALUE un lvalue puede ir a la derecha o izquierda de una asignación. Ejemplo:
  int a =100;
  int b;
  b=a
  a y **b** son lvalues, note que a **a** se le asigna un valor (a la izqierda de la asignación), pero también va a la derecha de la asignación en la última línea.
- RVALUE un rvalue, solo puede ir a la derecha de la asignación. En el ejemplo anterior 100 es un rvalue.
- Las variables on lvalues.

Operadores

Operadores artiméticos

Los operadores aritmeticos básicos son los siguientes, para int x=8, y=-3:

Operador	Descipción	Ejemplo
+	Adición	x+y es igual con 5
-	Substracción	x-y es igual con 11
*	Multiplicación	x*y es igual con -24
/	División	x/y es igual con -2 (enteros).
%	modulo	x%y es igual con 2
++	Operador de incremento por un entero	x++ es igual con 9.
–	Operador decremento por un entero	x– es igual con 7.

Se usan, *generalmente*, para operaciones aritméticas entre variables del mismo tipo.

Operadores relacionales

Los operadores relacionales básicos son los siguientes, para int x=8, y=-3:

Operador	Descipción	Ejemplo
==	igualdad	x==y es falso
!=	desigualdad	x!=y es verdadero
$>$	mayor que	$x > y$ es verdadero
$<$	menor que	$x < y$ es es falso.
$>=$	mayor o igual	$x >= (y + 11)$ es verdadero
$<=$	menor o igual	$(x - 12) <= y$ es verdadero

Se usan, generalmente, en expresiones dentro del condicional if o en ciclos while.

Operadores lógicos

Los operadores relacionales básicos son los siguientes, para int x=8, y=-3:

Operador	Descipción	Ejemplo
&&	and lógico	(x==y && x!=y) es falso, (x!=y && x>y) es verdadero
||	or lógico	(x==y || x!=y) es verdadero
!	not lógico	$!(x == y)$ es verdadero

Se usan, generalmente, para evaluar mas de una expresión en condicionales if o ciclos while.

Operadores miscelaneos

Los operadores relacionales básicos son los siguientes, para int x=8, y=-3:

Operador	Descipción	Ejemplo
sizeof()	Tamaño de una variable o tipo en bytes	sizeof(x), para x entera es 4
&	Devuelve la dirección de memoria de una variable	&x devuelve la direccion de la variabl x.
*	Indica una variable tipo apuntador	int *x; declara x como apuntador a entero.
*	Devuelve el contenido de una dirección de memoria puede ser lvalue o rvalue	int x=10; *(&x) es igual con diez.

Se usan, generalmente, en expresiones dentro del condicional if o en ciclos while.

printf y scanf

Impresión y lectura de variables

Con el fin de practicar con los tipos de datos, antes de ver funciones veremos como leer e imprimir variables de diferentes tipos con una función, por lo pronto lo tomaremos como receta, y después iremos viendo a detalle los temas.

- Para leer/imprimir necesitamos la libreria de entrada y salida estándar. $< stdio.h >$.
- Una función main que es la llamada por el sistema al momento de ejecutar el programa.
- Declaración de variables.
- Lectura, manipulación y escritura de la(s) variables.

```
1 #include <stdio.h>
  int main()
3 {
    int a, b;
5   scanf("%d %d",&a, &b);
    printf("Impresion de a=%d y b=%d", a,b);
7 return 0;
  }
```

Impresión y lectura de variables

La función printf imprime a la salida estándar (consola) el texto y variables que se le indiquen con un especificador de formato, la sintaxis es la siguiente:

printf(“Texto %formato \n”, variable);

- Primero va el nombre de la función y luego los argumentos entre paréntesis, y al final punto y coma para finalizar la instrucción (esto es lo mismo en todas las llamadas a función).
- Los argumentos van separados por comas (en todas las llamadas a función).
- El primer argumento es texto o una cadena constante de caracteres, que se tiene que poner entre comillas.
- El segundo argumento es la (lista de) variable(s).
- Por cada especificador de formato debe de haber una variable, de otra forma se imprime basura.

Ejemplos printf()

Los especificadores de formato son: %c char, %d (%i) int, %e (%E) formato exponencial, %f flotante, %g (%G) formato double en forma decimal o exponencial, puede usar %e o %f. %o octal (entero sin signo), %p apuntador, %s arreglo de caracteres, %u entero sin signo, %x (%X) valor hexadecimal (entero sin signo).

```
#include <stdio.h>
\int main()
{
   int a=0, b=1;
   double x,y;
   printf("Impresion de a=%d y b=%d", a,b);
   printf("Impresion de x=%g y y=%g", x,y);
return 0;
}
```

scanf

La funcion scanf es similar a printf pero para lectura desde la entrada estándar (consola). En general su uso es: scanf("%formato", &variable);

```
1 #include <stdio.h>
\int main()
3 {
    int a=0, b=1;
5   double x,y;
    scanf("%lf %lf", &x,&y);
7 return 0;
}
```

Note que se utiliza el operador &, esto quiere decir que los argumentdos (del segundo en adelante) de scanf son direcciones de memoria. En particular la dirección de memoria de donde se quiere almacenar el dato.

Resumen y Tareas

Resumen 1. ¿Qué debería de saber hasta aquí?

- Codificar, compilar y ejecutar un programa (sencillo sin funciones) en C.
- Evitar errores de overflow en enteros y flotantes.
- Leer, escribir en pantalla, realizar operaciones aritméticas con diferentes tipos de datos.
- No cometer errores de falta de inicialización, operación o asignación con diferentes tipos.
- Utilizar operadores aritméticos, lógicos y relacionales.

Tarea 1, programas 1-6

prog1.1 Escriba un programa que lea y escriba valores de los diferentes tipos de datos. ¿Que pasa si leo o escribo variables con un especificador de formato que no corresponde al tipo?, ¿ Qué pasa si uso mas especificadores de formato que variables o viceversa? Reporte sus observaciones (interesantes).

prog1.2 Escriba un programa que realice operaciones entre variables del mismo tipo y de diferentes tipos. ¿Que pasa si mezclo diferentes tipos de variables en la asignación u operación? Realice ejemplos, y reporte sus observaciones.

prog1.3 Escriba un programa que ejecute errores de integer overflow y floating point overflow.

prog1.4 Escriba un programa que ejemplifique la asignación del resultado de operadores no arítmeticos en variables enteras y comente su utilidad.

prog1.5 Escriba un programa que sume e imprima la cantidad de memoria de las variables declaradas en él. Utilice variables de todos los tipos básicos vistas en esta clase.

prog1.6 Investigue y programe ejemplos en un programa de los operadores: +=, -=, *=, /= y %=, con diferentes tipos de datos.

* Para todos los programas proveer ejemplos de entrada/salida usados/obtenidos mediante redirección. NO enviar ejecutables ni objetos, solo código.

Compilación y ejecución

Compilación desde consola

Considere el siguiente programa:

```
#include <stdio.h>

int main()
{

   int x=8, y=-3;
   scanf("%d %d",&x, &y);
   printf("res=%d %d\n",x,y);
   return 0;
}
```

Se compila el(los) archivos .c, como sigue:

gcc -c main.c
Se genera un objeto .o, y se compila el ejecutable:
gcc -o ejecutable main.o

Compilación de programas

Los programas deben de compilar en gcc. Para windows:

devC++ y codeblocks, entre otros, usan gcc.

En linux gcc es el compilador mas común y se puede usar como herramienta de consola, y con muchos IDEs y editores de código tambien.

En mac se que se puede instalar gcc, y se puede usar con las herramientas de mac, nunca lo he utilizado.

Redirección de la entrada estándar

Una forma de ejecutar un programa es pasarle los parámetros que lee con scanf redireccionandolos de un archivo, suponga que tiene el siguiente programa (izquierda) y archivo (derecha):

```
#include <stdio.h>
int main()
{

    int x=8, y=-3;
    scanf("%d %d",&x, &y);
    printf("res=%d %d\n",x,y);
    return 0;
}
```

datos.in es un archivo de texto que contiene lo siguiente:
3 -10

Y que el ejecutable se llama **ejecutable**, desde la carpeta que lo contiene en una consola, en linux:
./ejecutable <datos.in
En windows:
ejecutable <datos.in

Redirección salida estándar

De la misma forma la redirección de la salida estándar a un archivo de texto se puede realizar en linux:

./ejecutable >salida.out

En windows:

ejecutable >salida.out

Entrada y salida redireccionada:
Linux:
./ejecutable< datos.in >salida.out
Windows:
ejecutable< datos.in >salida.out

Errores

Errores comunes

Existen 3 tipos de errores (al menos) cuando se compila/ejecuta un programa:

- Errores de tiempo de compilación.
- Errores de linkeo o ligado.
- Errores de tiempo de ejecución.

Errores en tiempo de compilación

- No poner el punto y coma al final de una línea.
- Pasar el número y tipo correcto de argumentos a una función.
- Escribir mal el archivo de cabecera de funciones.
- Que todas las llaves { } y paréntesis se abran/cierren correctamente (que hagan juego).
- Usar variables no declaradas.
- etc.

Mensajes de error en tiempo de compilación

Cuando un error sucede en tiempo de compilación, generalmente por mala sintáxis, se muestra (por gcc en el IDE o consola) un error como el siguiente:

```
prog.c:  In function ‘main’:
```

Quiere decir que hay un error en el archivo `prog.c` en la función main
La siguiente línea normalmente indica el tipo y lugar preciso del error o warning:

```
prog.c:8:10:  warning:  unknown escape
sequence:  ’\ z’ [enabled by default]
```

El error está (aproximadamente) en la línea 8 y columna 10.
Y que el problema es que no conoce la secuencia de escape ’\ z’.

Errores de compilación

- 'variable' undeclared (first use in this function). Quiere decir que la variable que estás usando (en la línea aproximada que marca el error) no ha sido declarada, puede ser que no haya sido declarada, que te hayas equivocado en escribirla ya sea en la declaración o cuando se usa, o incluso un punto y coma al final de la declaración.
- parse error before 'XXXXX'. El compilador encuentra algo *irreconocible*, no es ni declaración de tipo ni llamada a función, y no sabe como procesarlo. Generalmente son errores de "dedazo".
- too few arguments to function 'YYYYY'. Se llama a una función y no se le pasan todos los argumentos necesarios.
- type mismatch with previous implicit declaration and previous implicit declaration of 'ZZZZZ'. Estás declarando dos veces la misma variable o función.
- parse error at end of input. Probablemente no cerraste una llave.
- unterminated string or character constant. No cerraste comillas o comilla simple.
- warning: passing arg n of 'XXXXX' makes pointer from integer without a cast. La función esperaba un apuntador y se le envía un entero, también está el error contrario, en ambos casos son de los más peligrosos y si se dejan pasar pueden llevar a corrupción de memoria.

Errores de tiempo de ejecución

Los más comúnes:

- Floating exception (core dumped). Error en una operación el valor devuelto se sale del rango de la variabble, división por cero, raíz cuadrada de un negativo.
- Segmentation fault (core dumped). Se accede a memoria que no ha sido requerida por el programa. Este error indica que el programa está mal, puede darse mucho cuando se trabaja con arreglos (y un indice se sale del rango o es negativo, etc.). O que se envíe una dirección de memoria incorrecta en una función. Un programa con un segmentation fault muchas veces tiene un error de lógica.

Referencias

Tipos de datos
http://www.tutorialspoint.com/cprogramming/c_data_types.htm
http://www.cs.uic.edu/~jbell/CourseNotes/C_Programming/DataTypes.html

Variables
http://www.tutorialspoint.com/cprogramming/c_variables.htm
http://en.wikibooks.org/wiki/C_Programming/Variables

printf y scanf
http://www.scenebeta.com/tutorial/entrada-y-salida-scanf-printf

Especificadores de formato:
http://www.le.ac.uk/users/rjm1/cotter/page_30.htm

Secuencias de escape:
http://www.le.ac.uk/users/rjm1/cotter/page_30.htm

Operadores
http://www.tutorialspoint.com/cprogramming/c_operators.htm
http://www.programiz.com/c-programming/c-operators

Errores comunes de compilacion
http://web.ics.purdue.edu/~cs240/misc/errors.html

Temas relacionados no cubiertos en está clase:

- Alcance de las variables.
- Macros
- Libreria limits.h, float.h con límites de variables.
- Todos los tipos posibles de datos básicos.
- Tipos de datos derivados en general.
- Tipos enum.
- Ver la declaración (sin definición de variables) con extern.

- Operadores a nivel bit.