***Cheat sheet: lenguaje C***

**Estructuras básicas de control**

* Secuencia
* Bifurcación
* Iteración

**Estructura general de un programa en C**

**#include <stdio.h>** Directivas del preprocesador

**#define EXIT 0** Macros del preprocesador

**float** num = 0; Variables globales

**void** message(); Prototipos de funciones

**int** main(){ Función main (declaraciones locales y sentencias)

printf(“Hello world!”);

message();

return EXIT;

}

**void** message(){ Definiciones de otras funciones

printf(“\nThis is a message.”);

}

**Elementos de un programa en C**

* **Tokens**:

Elementos léxicos de los programas.

* + *Identificadores*

Es una secuencia de caracteres letras, dígitos y subrayados (“\_”). El primer carácter debe ser letra. C diferencia entre mayúsculas y minúsculas. Pueden tener cualquier longitud, aunque sólo son significativos los primeros 32 caracteres. No pueden ser palabras reservadas. Ejemplos: Sueldo\_basico, Num\_Empleado, total45, Monto, monto, xy87z.

* + *Palabras reservadas (keywords o reserved words)*

Es un identificador asociado con una característica de significado especial. Una Palabra Reservada no se puede utilizar como nombre de un identificador o función del usuario. Ejemplos: for, while, if, sizeof, do, break, else, int, switch, const, case, struct.

* + *Literales*

También reciben el nombre de constantes. Son ciertos valores predefinidos (por el usuario o por la máquina) cuyo valor no se altera a lo largo del programa.

* + *Operadores*

Véase el apartado **Operadores y expresiones** en la página 7.

* + *Separadores*

Todas las sentencias (instrucciones) en C deben terminar con un punto y coma (;). Los separadores que se pueden usar en C son:

* + - Espacio en blanco (“ “)
    - Tabulador (“\t”)
    - Retorno de carro (CR) (“\r”)
    - Avance de línea (NL) (“\n”)
* **Comentarios**:

Se encierran entre /\* … \*/ y pueden extenderse a lo largo de varias líneas. Ejemplos:

// Programa para calcular el promedio de un grupo de números.

/\* Nombre del archivo: media.c \*/

* **Sentencia para el preprocesador**:

Son órdenes que el preprocesador interpreta antes de que el código fuente sea compilado.

* **Archivos de encabezado**:

Es un archivo especial que contiene declaraciones de elementos y funciones de la biblioteca.

Para utilizar macros, constantes, tipos y funciones almacenadas en una biblioteca, un programa debe utilizar la directiva **#include** para insertar el archivo de encabezado correspondiente. Por ejemplo:

**#include <stdio.h>**

**#include “stdio.h”**

**Tipos de datos**

C no soporta un gran número de tipos de datos predefinidos, pero tiene la capacidad para crear sus propios tipos de datos. Todos los tipos de datos simples o básicos de C son, esencialmente, números. Los tres tipos de datos básicos son:

* *Enteros*

Números completos y sus negativos, de tipo **int**. Sus variantes son los tipos **short**, **long** y **unsigned**. Representa un subconjunto finito de  números enteros. El número mayor que puede representar depende del tamaño del espacio usado por el dato y la posibilidad (o no) de representar números negativos. El tamaño del tipo de dato entero es de 4 Bytes = 4 x 8 = 32 bits. Con 32 bits se pueden representar valores.

* *Números de punto flotante (reales)*

Números decimales, tipos **float**, **double** o **long double**. Es un subconjunto finito de los números racionales con el que se pretende representar al conjunto infinito de los números reales. Se utilizan cuando se calculan funciones que requieren precisión fraccionaria. Los tipos de datos de punto flotante representan números reales que contienen un punto decimal, tal como 3.14159, o números muy grandes, tales como , el tamaño del dato depende del tipo de número real que se declare.

* *Caracteres*

Letras, dígitos, símbolos y signos de puntuación, tipo **char**.

**Tabla de tipos de datos en C**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo | Tamaño en bits | Dígitos de precisión | Rango | |
| **Min** | **Max** |
| bool | 8 | 0 | 0 | 1 |
| char | 8 | 2 | -128 | 127 |
| unsigned char | 8 | 2 | 0 | 255 |
| short int | 16 | 4 | -32,768 | 32,767 |
| unsigned short int | 16 | 4 | 0 | 65,535 |
| int | 32 | 9 | -2,147,483,648 | 2,147,483,647 |
| unsigned int | 32 | 9 | 0 | 4,294,967,295 |
| long int | 32 | 9 | -2,147,483,648 | 2,147,483,647 |
| unsigned long int | 32 | 9 | 0 | 4,294,967,295 |
| long long int | 64 | 18 | -9,223,372,036,854,775,808 | 9,223,372,036,854,775,807 |
| unsigned long long int | 64 | 18 | 0 | 18,446,744,073,709,551,615 |
| float | 32 | 6 | -3.4e38 | 3.4e38 |
| double | 64 | 15 | -1.7e308 | 1.7e308 |
| long double | 96 | 18 | -1.19e4932 | 1.19e4932 |

**Definiendo una constante**

Las constantes pueden recibir nombres simbólicos mediante la directiva **#define**:

**#define NL ‘\n’**

**#define dato 5**

**#define PI 3.141592**

**#define VALOR 77**

C sustituye los valores \n, 5, 3.141592 y 77 cuando se encuentra las constantes simbólicas NL, dato, PI y VALOR. Los ejemplos anteriores no son sentencias y, por ello, no terminan en punto y coma.

printf("El valor es %c, %d ",NL,VALOR);

Otra manera de declarar una constante es agregando const a la declaración de una variable:

const tipo nombreConstante = valor;

**const float** PI = 3.1416;

**Constantes enteras**

La escritura de constantes enteras requiere seguir unas determinadas reglas:

* No utilizar nunca punto ni otros signos de puntuación en números enteros o completos. Por ejemplo: 123456 (no 123.456)
* Para forzar un valor al tipo long, terminar con una letra L o l. Por ejemplo: 1024 es un tipo entero; 1024L, es un tipo largo (long)
* Para forzar un valor al tipo unsigned, terminarlo con una letra mayúscula u. Por ejemplo: 4352U.

**Constantes reales**

Una constante flotante representa un número real; siempre tienen signo y representan aproximaciones en lugar de valores exactos. Ejemplos:

82.347 0. 63 83.0 47e-4 1.25E7 6l.e+4

La notación científica se representa con un exponente positivo o negativo.

2.5E4 equivale a 25000

5.435E-3 equivale a 0.005435

Existen tres tipos de constantes:

* float 4 bytes
* double 8 bytes
* long double 10 bytes

**Constantes carácter**

Una constante carácter (char) es un carácter encerrado entre apóstrofes. Por ejemplo: ‘A’, ‘b’, ‘C’.

Además de los caracteres ASCII estándar, una constante carácter soporta caracteres especiales que no se pueden representar utilizando su teclado, como, por ejemplo, los códigos ASCII altos y las secuencias de escape. Así, por ejemplo, el carácter arroba (@) -código ASCII unicode, hex 40- se representa mediante el prefijo \x y el número hexadecimal del código ASCII. Por ejemplo,

**char** arroba = '\x 40';

Este método se utiliza para almacenar o imprimir cualquier carácter de la tabla ASCII por su número hexadecimal. En el ejemplo anterior, la variable arroba no contiene cuatro caracteres sino únicamente el símbolo arroba (‘@’).

**Constantes literales**

Una constante cadena (también llamada literal cadena o simplemente cadena) es una secuencia de caracteres encerrados entre comillas (“ ”). Algunos ejemplos de constantes de cadena son:

"123"

"12 de octubre de 1492"

"esto es una cadena"

Se puede escribir una cadena en varias líneas, terminando cada línea con ‘’\”, por ejemplo:

"esto es una cadena\

que tiene dos lineas"

Se puede concatenar cadenas, escribiendo “ABC” “DEF” “GHI” “JLK” que equivale a “ABCDEFGHIJKL".

**Variables**

En C una variable es una posición de memoria con un nombre asignado donde se almacena un valor de un tipo de dato. Su valor puede cambiar a lo largo de la ejecución del programa.

La forma más simple de una declaración de variable en C es poner primero el tipo de dato y a continuación el nombre de la variable. Si se desea dar un valor inicial a la variable, éste se pone a continuación. El formato de la declaración es:

<tipo de dato> <nombre de variable > = <valorinicial>

Se pueden también declarar múltiples variables en la misma línea:

<tipo de dato> <nom-var1, nom-var2> . . . <nom-varn>

Por ejemplo:

**int** longitud;

**int** valor = 99;

**int** valorl , valor2;

**int** num-parte = 1141, num-items = 45;

**Funciones básicas de entrada y salida**

* *Entrada*

Se realiza mediante la función scanf(). Esta función captura los datos introducidos a través del teclado. Tiene la siguiente sintaxis:

scanf(cadena\_de\_control, var1, var2, … var n);

donde cadena\_de\_control contiene los tipos de dato y “var1, var2, … var n” son direcciones de las variables del tipo de los códigos de control; cada una debe ser un apuntador. Ejemplos:

scanf(“%d”,&edad);

scanf(“%f %f”,&x1,&y1);

A continuación una tabla con las cadenas de control que se pueden utilizar:

|  |  |
| --- | --- |
| Formato | Descripción |
| %d | El dato se convierte a entero decimal |
| %o | El dato entero se convierte a octal |
| %x | El dato entero se convierte a hexadecimal |
| %u | El dato entero se convierte a entero sin signo |
| %c | El dato se considera de tipo carácter |
| %e | El dato se considera de tipo float. Se convierte a notación científica. |
| *%f* | El dato se considera de tipo float. Se convierte a notación decimal, con parte entera y los dígitos de precisión. |
| %g | El dato se considera de tipo float. Se convierte según el código *%e o %f* dependiendo de cuál sea la representación más corta. |
| %s | El dato ha de ser una cadena de caracteres. |
| %lf | El dato se considera de tipo double. |

Algunas funciones simples que son de ayuda a la hora de leer cadenas y caracteres son las funciones getchar() y fgets(), primero incluyendo la biblioteca **<stdlib.h>**.

* *Salida*

Se realiza mediante la función printf(). Esta función visualiza en pantalla los datos, transforma los datos, que están en representación binaria, a ASCII. Tiene la siguiente sintaxis:

printf(cadena\_de\_control, dato1, dato2, … dato\_n);

donde cadena\_de\_control contiene los tipos de dato y su forma de mostrarlos, mientras que “dato1, dato2, … dato\_n” pueden ser variables, constantes, o en general expresiones de salida. Ejemplos:

printf(“\nSu edad es: %d”, edad);

printf(“\nLa distancia entre los puntos es: %f 12.4 \n”, dist);

printf(“\t Punto 1 \t \t \t Punto 2 \t Distancia \n\a”);

printf(“\t (x1= %8.2f , \t y1= %8.2f) \t \t (x2=%8.2f, \t y2=%8.2f) \t %8.2f \n”, x1, y1, x2, y2, distancia);

A continuación una tabla con los códigos de escape más utilesque se pueden utilizar:

|  |  |  |  |
| --- | --- | --- | --- |
| Código | Significado | Códigos ASCII | |
| Dec | Hex |
| \n | Nueva línea | 10 | 0A |
| \r | Retorno de carro | 13 | 0D |
| \t | Tabulación | 9 | 09 |
| \v | Tabulación vertical | 11 | 0B |
| \a | Pitido sonoro | 7 | 07 |
| \b | Retroceso de espacio | 8 | 08 |
| \f | Avance de página | 12 | 0C |
| \\ | Barra inclinada inversa | 92 | 5C |
| \’ | Comilla simple | 39 | 27 |
| \” | Comillas dobles | 34 | 22 |
| \? | Signo de interrogación | 63 | 3F |
| \000 | Número octal | Todos | Todos |
| \xhh | Número hexadecimal | Todos | Todos |

Algunas funciones simples que son de ayuda a la hora de imprimir cadenas y caracteres son las funciones putchar() y puts(), primero incluyendo la biblioteca **<stdlib.h>**.

**Operadores y expresiones**

Una expresión es una sucesión de operadores y operandos debidamente relacionados para formar expresiones matemáticas que especifiquen un cálculo. Observemos los operadores más fundamentales en C:

* *Operadores de asignación*

Tienen la forma **variable = expresión**, donde **variable** es cualquier identificador. La asociatividad es por la derecha. Ejemplo:

a = b = c = 10

Los operadores de asignación en C son:

|  |  |  |
| --- | --- | --- |
| Símbolo | Uso | Descripción |
| = | a = b | Asigna el valor de b a a |
| += | a += b | Equivalente a escribir a = a + b |
| -= | a -= b | Equivalente a escribir a = a - b |
| \*= | a \*= b | Equivalente a escribir a = a \* b |
| /= | a /= b | Equivalente a escribir a = a / b |
| %= | a %= b | Equivalente a escribir a = a % b |

* *Operadores aritméticos*

Los operadores aritméticos sirven para realizar operaciones aritméticas básicas. Los operadores aritméticos C siguen las reglas algebraicas típicas de jerarquía o prioridad. Los operadores aritméticos en C son:

|  |  |  |  |
| --- | --- | --- | --- |
| Símbolo | Tipos enteros | Tipos reales | Ejemplo |
| + | Suma | Suma | 5 + a |
| - | Resta | Resta | 6 – x |
| \* | Producto | Producto | x \* y |
| / | División entera: Cociente | División en coma flotante | y / 3 |
| % | División entera: Resto |  | 5 % 2 |

* *Operadores de incremento y decremento*

Los operadores ++ y --, denominados de incremento y decremento, suman o restan 1 a su argumento, respectivamente, cada vez que se aplican a una variable.

|  |  |
| --- | --- |
| Incremento | Decremento |
| n++ | n-- |
| n += 1 | n -= 1 |
| n = n + 1 | n = n - 1 |

* *Operadores relacionales*

C no tiene tipos de datos lógicos o booleanos, para representar los valores verdadero (true) y falso (false) se utiliza el tipo int, con el valor entero 0 que representa a **falso** y distinto de cero a **verdadero**. Tienen la sintaxis **expresión1** *operador\_relacional* **expresión2**

|  |  |  |
| --- | --- | --- |
| Operador | Significado | Ejemplo |
| == | Igual a | a == b |
| != | Distinto de | a != b |
| > | Mayor que | a > b |
| < | Menor que | a < b |
| >= | Mayor o igual que | a >= b |
| <= | Menor o igual que | a <= b |