UAEM | Universidad Autónoma del Estado de México

UNIDAD ACADÉMICA PROFESIONAL TIANGUISTENCO

LICENCIATURA EN INGENIERÍA DE SOFTWARE

UNIDAD DE APRENDIZAJE: PROGRAMACIÓN

Créditos institucionales de la UA: 8
Material visual: Diapositivas

Unidad de competencia II

INTRODUCCIÓN A LA PROGRAMACIÓN ESTRUCTURADA

Elaborado por M. en C. Selene Palacios Astudillo

Período 2015-A

¿Cómo emplear este material?

El presente material tiene como finalidad facilitar la exposición gráfica del tema “Introducción a la Programación Estructurada” que se aborda en la unidad de aprendizaje “Programación” que corresponde al segundo semestre de la Licenciatura en Ingeniería de Software.

La presentación deberá ir acompañada de una explicación oral del docente, ya que la aportación que pueda hacer mediante ejemplos y situaciones cotidianas brindará la oportunidad de que los estudiantes comprendan la importancia de construir argumentos sólidos, creíbles y bien soportados.

INTRODUCCIÓN A LA PROGRAMACIÓN ESTRUCTURADA

- ÍNDICE -

INTRODUCCIÓN A LA PROGRAMACIÓN ESTRUCTURADA

Objetivo de la Unidad Temática.

Al término de la unidad temática, los estudiantes deberán conocer los conceptos básicos de la programación estructurada, tales como identificadores, tipos de datos, operadores y expresiones. Así como las funciones que permiten la transferencia de información entre la computadora, los dispositivos de entrada/salida estándar, la estructura básica de un programa y desarrollar los programas básicos.

- SUB TEMAS -

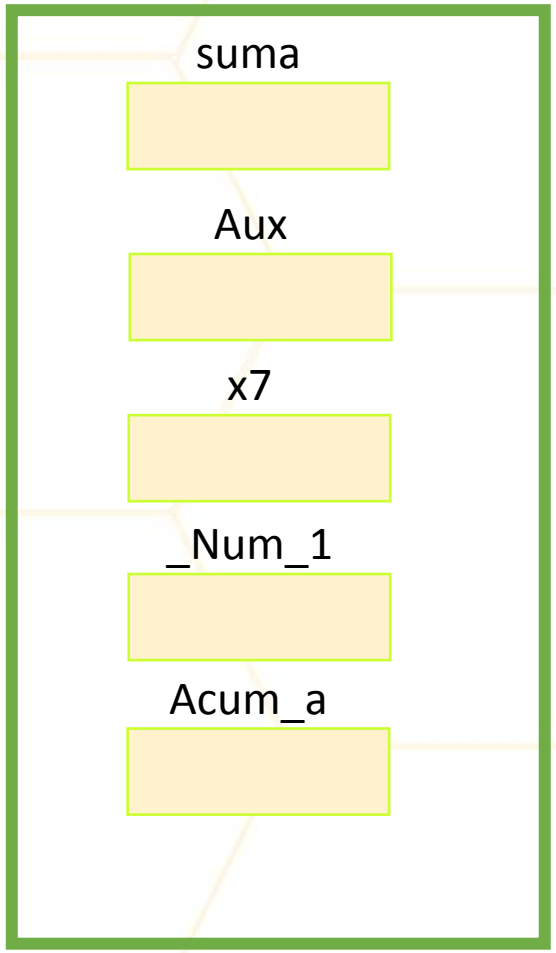
Identificadores, tipos de datos, operadores y expresiones.

- Identificadores: variables y constantes.
- Tipos de datos simples.
- Operandos, operadores y precedencia.
- Construcción de expresiones.
- Sentencia y bloques de sentencias de programa.

Índice

Identificadores

Nombre que se asigna a las casillas o celdas de memoria en una computadora, para su posterior utilización.



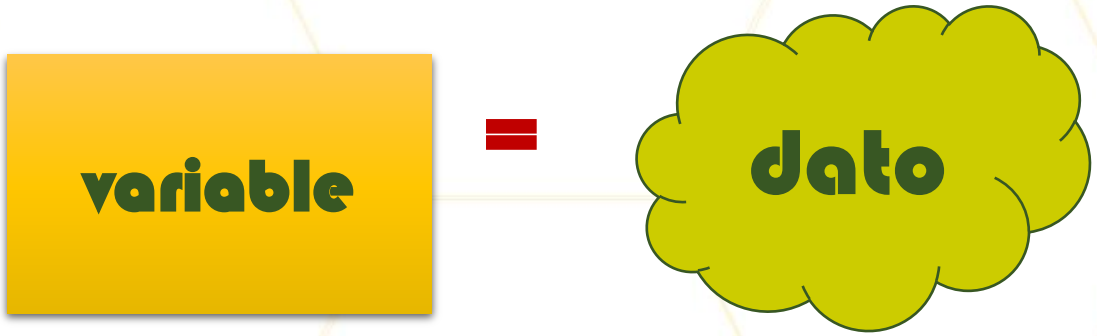
Identificadores - Reglas -

- Sentive case.
- Puede iniciar con un caracter alfabético ó _.
- Del segundo caracter en adelante, puede ser caracter alfabético, numérico o _.
- No caracteres especiales.
- No palabras reservadas.
- No espacio en blanco.
- Nombres representativos a su función.

Identificadores - Variables -

contenedor

contenido

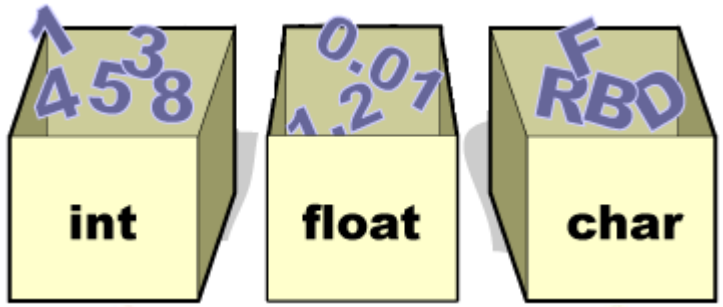


- ✓Espacio de memoria, que puede cambiar su valor durante la ejecución del programa.
- ✓Limpiarlas antes de su utilización.
- ✓Su tipo será de acuerdo al valor que almacenarán.
- ✓Deben ser declaradas antes de utilizarse.

Identificadores - Constantes -

contenedor **contenido**

variable = Valor Constante



Identificadores - Variables y Constantes -

Sintaxis:

<TipoDeDato> <NombreDeVariable>;

<TipoDeDato> <ListaDeVariables>;

<TipoDeDato> <NombreDeVariable>=<valor>;

const<TipoDeDato> <NombreDeVariable>=<valor>;

#define<NombreConstante><valor>

```
int x;

char x,y,z;

int x=10;

const  float salario =512.36;

#define pi 3.1415
```

Tipos de datos

Tipo	Tamaño (bytes)	Rango min	Rango max
char	1	0	255
int	4	-2 147 483 648	2 147 483 647
float	4	$1.17549*(10^{-38})$	$3.40282*(10^{38})$
double	8	$2.22507*(10^{-308})$	$1.79769*(10^{308})$
short int	2	-32 768	32 767
unsigned int	4	0	4 294 967 295
long int	4	-2 147 483 648	2 147 483 647
long double	12	$3.3621*(10^{-4932})$	$1.18973*(10^{4932})$

Tipos de datos

Tipo	Tamaño (bytes)	Rango min	Rango max
bool	1	0	1
unsigned long long int	8	0	18 446 744 073 709 551 615
long long int	8	+/- 9 223 372 036 854 775 808 – 7	
unsigned long int	4	0	4 294 967 295
unsigned short int	2	0	65 535
signed char	1	-128	127

Operandos, operadores y precedencia

$$z = p * r \% q + w / x - y$$

6 1 2 4 3 5

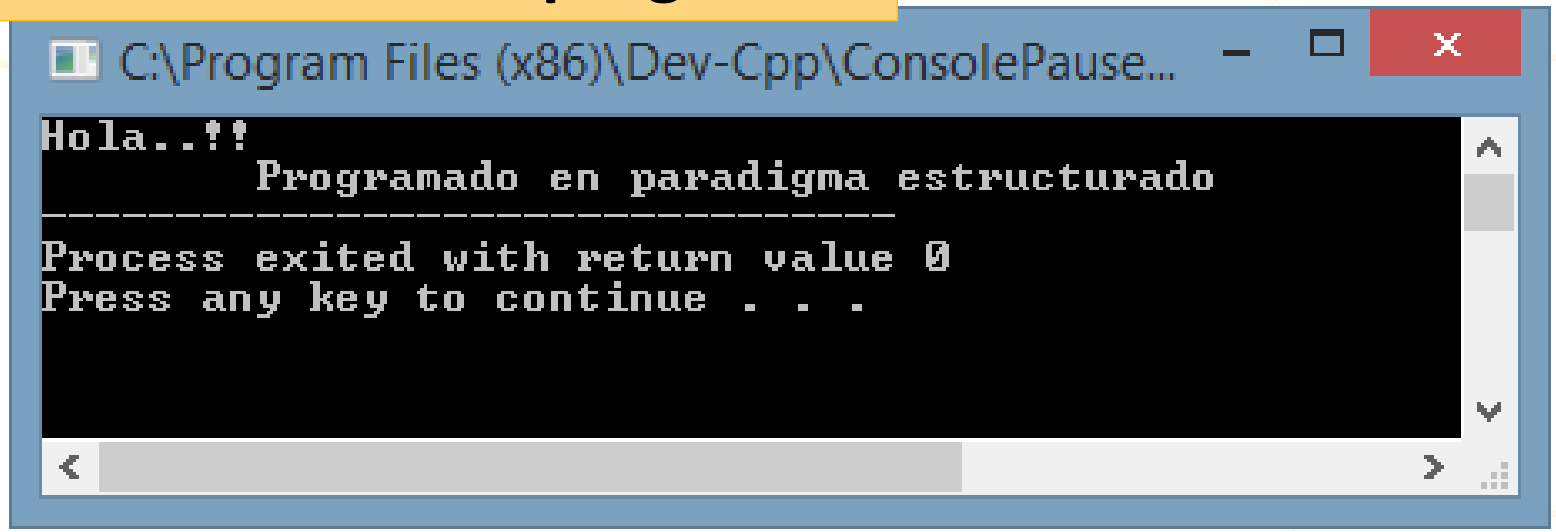
Operandos, operadores y precedencia

Aritméticos	Relacionales	Lógicos	Asignación	Incremento/Decremento
+	>	&&	=	++
-	<	||	+=	--
*	>=	!	-=	
/	<=		*=	
%	==		/=	
	!=		%=	

Operandos, operadores y precedencia

( ) [ ]
+ - ++ -- !
* / %
* -
> >= < >=
== !=
&&
||
= += -= *= /= %=

Sentencia y bloques de sentencia de un programa.




```c
/*Programa que muestra un
mensaje  en la consola*/

#include <stdio.h> //biblioteca
int main()
{
    printf("Hola..!!\n");
    printf("\tProgramado en paradigma estructurado");
return 0;
}
```

- SUB TEMAS -

Salida con formato.

- Especificadores de formato.
- Secuencias de escape.
- Ancho de campo.

Índice

Especificadores de formatos

printf (“print formatted”).

- ✓ **Imprime un mensaje por pantalla.**
- ✓ **Utiliza “una cadena de formato”.**
- ✓ **Incluye las instrucciones para mezclar múltiples cadenas en la cadena final a mostrar por pantalla.**
- ✓ **Incluye texto y “marcas” a reemplazar por texto.**
- ✓ **Recibe un número variable de parámetros.**
- ✓ **El símbolo “%” denota el comienzo de la marca de formato.**
- ✓ **Si en la cadena de formato aparecen varias marcas, los valores a incluir se toman en el orden en el que aparecen.**

Especificadores de formatos

Carácter	Formato
%i %d %u	signed, , unsigned
%o	
%x %X	
%f	
%e %E	
%g %G	e ó f corto
%c	
%s	
%p	

Secuencias de escape

Carácter	Significado
\n	Nueva línea
\t	Tabulador H
\’	Comilla
\”	Comilla
\?	Signo
\\	Barra invertida
\a	Alerta
\b	Retroceso (BS)
\v	Tabulador V
\f	Avance página
\r	Retorno

Ancho de campo

```
%[parameter][flags][width][.precision][length]type
```

```
printf("\nColor %s, Numero %d, Real %5.2f",  "rojo" ,1234567, 3.1416);
```

```
Color rojo, Numero 1234567, Real  3.14
```

```
printf("\n%.*s",3,"abcdef");
```

```
abc
```

```
printf("\n%*d", 5, 110);
```

```
  110
```

Ejemplo.

Determina que realizan las siguientes sentencias:

a) printf (“Hola”);

b) printf(“\nEl numero %d es par\n”, n);

c) printf(“Los datos son %c %d %%%f \n”, ‘a’, 28, 3.4);

- SUB TEMAS -

Entrada con formato.

- Entrada con formato.

- Índice

Entrada con formato

```
#include <stdio.h>
int scanf(const char *format,...);
```

scanf().

- ✓ **Permite leer varios tipos de datos.**
- ✓ **Se pueden especificar varios especificadores de formato en la variable de tipo puntero *format*.**
- ✓ **Necesita pasar punteros a los argumentos, para que pueda modificar sus valores.**

Entrada con formato

```
scanf("%d %d", &entero1, &entero2);

scanf("%f", &decimal);

scanf("%s", cadena);
```

Entrada con formato

```c
#include <stdio.h>

#define TAM_MAXIMO 80

int main(void)
{
    char cadena[TAM_MAXIMO];
    int entero1, entero2;
    float decimal;

    printf("\nIntroduce dos enteros separados por enter: ");
    scanf("%d %d", &entero1, &entero2);
    printf("\nIntroduce un numero decimal: ");
    scanf("%f", &decimal);
    printf("\nIntroduce una cadena: ");
    scanf("%s", cadena);
    printf("\nEsto es lo que has escrito:\n");
    printf("%d\n%d\n%5.2f\n%s \n", entero1, entero2, decimal, cadena);
  return 0;
}
```

Entrada con formato

```
printf("\nIntroduce dos enteros separados por enter: ");
scanf("%d %d", &entero1, &entero2);
```

```
Introduce dos enteros separados por enter: 5
147
```

```
printf("\nIntroduce un numero decimal: ");
scanf("%f", &decimal);
```

```
Introduce un numero decimal: 123.123
```

```
printf("\nIntroduce una cadena: ");
scanf("%s", cadena);
```

```
Introduce una cadena: Bienvenido
```

Entrada con formato

```
printf("\nEsto es lo que has escrito:\n");
printf("%d\n%d\n%5.2f\n%s \n", entero1, entero2, decimal, cadena);
```

```
Esto es lo que has escrito:
5
147
123.12
Bienvenido
```

Ejemplo:

Adapte las siguientes sentencias, en un programa C, para leer los datos, según el formato de entrada:

a) scanf (“%f”, &numerof);

b) scanf (“%c”, &letra);

c) scanf(“%s %d %c”, cadena, &n, &letra);

d) scanf (“%s”, cadena);

- SUB TEMAS -

Otras funciones de Entrada y Salida.

❑ Índice

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
int getc(FILE *stream);
```

getc().

- ✓ Lee el siguiente carácter de un flujo de fichero y devuelve el valor numérico de ese carácter.
- ✓ ***stream***, es una variable con el flujo de fichero, si llega al final del fichero o hay algún error, la función devuelve EOF.

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
int getchar(void);
```

getchar().

- ✓ Lee el siguiente carácter de un flujo de fichero y devuelve el valor numérico de ese carácter.
- ✓ void, ningún argumento para llamar a la función.

Otras funciones de Entrada y Salida.

```c
#include <stdio.h>

int main(void)
{
    int caracter1;
    char caracter2;
    printf("\nTeclee, dos caracteres seguidos: ");
    caracter1=getc(stdin);
    caracter2=getchar();
    printf("\nEl primer caracter que has introducido es %c ", caracter1);
    printf("\nEl segundo caracter que has introducido es %c ", caracter2);
    return 0;
}
```

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
int putc(int c, FILE *stream);
```

putc().

- ✓ Escribe un carácter al flujo del fichero especificado.
- ✓ Si la operación tiene algún erros, devuelve EOF, si no, devuelve el carácter escrito.

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
int putchar(int c);
```

putchar().

✓Escribe un carácter en la pantalla; usa siempre la salida estándar que este predefinida.

Otras funciones de Entrada y Salida.

```c
#include <stdio.h>

int main(void)
{
   int caracter1 = 65; /*Suele ser el valor numérico de A*/
   char caracter2 = 'A';

   printf("\nLa letra con valor numerico de 65 es: ");
   putc(caracter1,stdout);
   printf("\nY la variable caracter2 contiene la letra: ");
   putchar(caracter2);
   return 0;
}
```

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
char *gets(char *s);
```

gets().

- ✓Lee una línea completa de caracteres.
- ✓Los caracteres leídos se guardan en el *array s.*
- ✓La función añade el carácter de terminación '\0', cuando encuentra el carácter de nueva línea o el de fin de fichreo EOF.
- ✓Si en la lectura hay algún error, devuelve un puntero a NULL.

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
int *puts(const char *s);
```

puts().

- ✓ **Escribe una secuencia de caracteres al flujo de salida estándar.**
- ✓ ***s*, se refiere al array que contiene la cadena de caracteres.**
- ✓ **Si la función se realiza correctamente devuelve cero. Si no, devuelve algo distinto de cero.**

Otras funciones de Entrada y Salida.

```c
#include <stdio.h>

#define TAM_MAXIMO 80

int main(void)
{
  char cadena[TAM_MAXIMO];

  printf("\nPor favor, escribe una línea de no más de 80 caracteres: ");
  gets(cadena);
  printf("\nLa línea que has introducido es: ");
  puts(cadena);
  return 0;
}
```

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
char *fgets(char *s, int n, FILE *stream);
```

fgets().

- ✓Lee cadenas introducidas por el usuario.
- ✓*s* referencia a un array de caracteres, almacena lo que lee de un fichero apuntado por *stream.*
- ✓*n* especifica el número máximo de elementos del array.
- ✓La función lee hasta n-1 caracteres, y añade el carácter nulo.
- ✓Si no ha llegado a n-1 caracteres, pero lee '\n', termina de leer y devuelve la cadena.
- ✓Si no hay errores la función devuelve el puntero *s*
- ✓Si encuentra EOF o existe algun error, devuelve NULL.

Otras funciones de Entrada y Salida.

```
#include <stdio.h>
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

getline().

- ✓Lee una línea entera de stream almacenando el texto en un buffer.
- ✓Almacena la dirección del buffer de *n bytes de largo usando malloc en*lineptr.
- ✓Si buffer no es suficiente para albergar la frase leída, getline hace el buffer más grande usando realloc, actualizando la nueva dirección en *lineptr y el tamaño en *n.
- ✓Si se inicializa *lineptr =null y *n=0, getline hace ese malloc.
- ✓Si todo es correcto, *lineptr es un char * que apunta al texto que se ha leído. La función devuelve el número de caracteres leídos sin contar el carácter '\0'. Si hay error o se alcanza final de fichero, devuelve -1 .

Otras funciones de Entrada y Salida.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void)
{
  ssize_t bytes_leidos;
  size_t numero_bytes;
  //ssize_t y size_t son sinónimos de unsigned int
  char *cadena;
  puts("Por favor, introduce una línea de texto:\n");
  numero_bytes = 0;
  cadena = NULL;
  bytes_leidos = getline(&cadena, &numero_bytes, stdin);
  if (bytes_leidos == -1)
  {
    puts("Error.");
  }
  else
  {
    puts("La línea es:");
    puts(cadena);
  }
  free(cadena);
  return 0;
}
```

Bibliografía

- Cairo Osvaldo y Guardati Silvia. Metodología de la Programación. Algoritmos, diagramas de flujo y programas. Alfa Omega, 2005. México.
- Ceballos Sierra Francisco Javier. Enciclopedia del lenguaje C. Alfa Omega, 2007. México.
- H. M. Deitel, P.J. Deitel. Como programar en C y C++ (2ª edición). Prentice Hall, 1995. México.
- Joyanes Aguilar, Luis. Programación en C++. Algoritmos, estructuras de datos y objetos (3ª edición). McGraw-Hill, 20063. España.
- Joyanes Aguilar, Luis. Fundamentos de programación. Libro de problemas (2ª edición). McGraw-Hill, 2003. España.

❑ Índice