

Introducción a la programación orientada a objetos (IPOO)

Expresiones, variables y tipos. Funciones y paso de
parámetros.

carlos.andres.delgado@correounivalle.edu.co

Carlos Andrés Delgado S.

Facultad de Ingeniería. Universidad del Valle

Febrero de 2017

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables
- 5 Strings
- 6 Contenedores STL
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios

Definición

Los lenguajes de programación, cuentan con expresiones para facilitar el trabajo a los programadores abreviando operaciones largas en un sólo paso, a esto se le conoce como **azúcar sintáctico**, por ejemplo:

```
int numero = 5;
int resultado = 3;

resultado = numero + resultado;
cout << resultado << endl;

resultado = 3;

resultado += numero;
cout << resultado << endl;
```

En este caso se han abreviado la operación de suma. Este método aplica a cualquier operación matemática.

Definición

También es posible definir que se realiza primero, si se muestra el valor o se realiza la operación. Un caso de esto son los operadores incremento o decremento.

```
++i // Pre-Incremento  
i++ // Incremento  
--i // Pre-decremento  
i-- // Decremento
```

Si los operadores que aparecen antes de las variables, primero se aplica la operación y luego se produce el valor actual de la variable. En el caso contrario, primero se produce el resultado y luego se muestra la variable.

Definición

Implemente:

```
int numeroA = 0;

cout << numeroA++ << endl;
cout << numeroA << endl;

int numeroB = 0;

cout << ++numeroB << endl;
cout << numeroB << endl;
```

¿Que diferencia observa?.

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables
- 5 Strings
- 6 Contenedores STL
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios

Variables

Una variable es un espacio de memoria administrado por un programa. En los lenguajes de programación consta de dos partes:

- 1 **Identificador:** El cual es un nombre simbólico
- 2 **Valor:** Información contenida en la variable.

Tipos

Los tipos de datos son la forma de indicarle a la máquina cómo almacenar la información. Es de anotar que la máquina sólo almacena 0 y 1, por lo que es necesario organizar esta información para su procesamiento. Un tipo de dato permite:

- 1 Especificar los bits que se requiere un tipo de dato
- 2 Especificar el significado del tipo de dato. ¿Es un número? ¿Una cadena? ¿Una lista?.

Lenguajes y tipos

Debido a que los lenguajes de programación presentan sus particularidades, estos se han clasificado en:

- 1 Dinámicamente y estáticamente tipados
- 2 Fuerte y débilmente tipados

Lenguajes y tipos

- 1 Estáticamente tipado:** En estos lenguajes es obligatorio especificar el tipo de cada variable. En otras palabras el tipo de las variables se define en la creación del programa. Ejemplo Java, C# y C++
- 2 Dinámicamente tipado:** En estos lenguajes es el contenido el que determina el tipo de variable. Es decir, el tipo de la variable se determina durante la ejecución. Ejemplo Python

Lenguajes y tipos

- 1 **Fuertemente tipado:** En estos lenguajes de programación el tipo de la variable no puede cambiar durante la ejecución. Ejemplo Java, C# y C++.
- 2 **Débilmente tipado:** El tipo de la variable puede cambiar durante la ejecución. Ejemplo PHP y Python

Tipos en C++

Para el caso de los números enteros se utilizan los siguientes tipos de datos:

Tipo	Tamaño bits	Mínimo	Máximo
Bool	8	0	1
char	8	-128	127
int	32	-2^{31}	$2^{31} - 1$

Tipos en C++

Para el caso de los números en punto flotante (o reales) se utilizan los siguientes tipos de datos:

Tipo	Tamaño bits	Mínimo	Máximo
float	32	$-1,175 \times 10^{38}$	$3,4 \times 10^{38}$
double	64	$\pm 2,22 \times 10^{-308}$	$\pm 1,79 \times 10^{308}$

Especificadores en C++

Los especificadores en C++, permiten modificar los valores máximo y mínimo de los tipos de datos. Los especificador son:

- 1 **De signo:** unsigned, para sólo trabajar números enteros
- 2 **De tamaño de representación en bits:** Para reducir o aumentar el número de bits en la representación binaria.

Definición

```
char c;  
unsigned char cu;  
int i;  
unsigned int iu;  
short int is;  
unsigned short int isu;  
long int il;  
unsigned long int ilu;  
float f;  
double d;  
long double ld;
```

Definición

```
cout << "char= " << sizeof(c) << endl;
cout << "unsigned char = " << sizeof(cu) << endl;
cout << "int = " << sizeof(i) << endl;
cout << "unsigned int = " << sizeof(iu) << endl;
cout << "short = " << sizeof(is) << endl;
cout << " unsigned short = " << sizeof(isu) << endl;
cout << "long = " << sizeof(il) << endl;
cout << "unsigned long = " << sizeof(ilu) << endl;
cout << " float = " << sizeof(f) << endl;
cout << " double = " << sizeof(d) << endl;
cout << " long double = " << sizeof(ld) << endl;
```

¿Que observa?. El comando **sizeof** permite conocer el tamaño que ocupa en memoria una variable, su salida está dada en bytes. **1 byte = 8 bits.**

Definición

Miremos la capacidad que nos proporcionan estos modificadores.

Tipo	# bits	Mínimo	Máximo
unsigned int	32	0	$2^{32} - 1$
short int	16	-2^{15}	$2^{15} - 1$
unsigned short int	16	-0	$2^{16} - 1$
long int	64	-2^{63}	$2^{63} - 1$
unsigned long int	64	0	$2^{64} - 1$
long double	128	$\pm 3,4 \times 10^{-4932}$	$\pm 1,1 \times 10^{4932}$

Apuntes sobre tipos

Cuando se trabaja en cualquier lenguaje de programación, se debe tener cuidado en la elección del tipo de datos. Existen dos problemas potenciales que deben considerarse:

- 1 Imprecisión:** El tipo de dato no puede representar todos los números en un rango o el resultado de una operación no es el esperado.
- 2 Desbordamiento:** El valor asignado a una variable no puede ser representado

Imprecisión

Este problema se presenta en las representaciones de números reales, los cuales son **float** y **double**. Pruebe lo siguiente:

```
//Por defecto cout muestra 4 cifras, aumentamos a 20
cout.precision(20);
float a= 10000.000977;
cout << a << endl;

double b= 10000.000977;
cout << b << endl;
```

¿Que observa?

Imprecisión

Este problema se presenta en las representaciones de números reales, los cuales son **float** y **double**. Pruebe lo siguiente:

```
//Por defecto cout muestra 4 cifras, aumentamos a 20
cout.precision(20);
float salarioFloat = 17.0;
for(int i=0; i<16; i++){
    salarioFloat+=0.001;
}
cout<<salarioFloat<<endl;

double salarioDouble = 17.0;
for(int i=0; i<16; i++){
    salarioDouble+=0.001;
}
cout<<salarioDouble<<endl;
```

¿Que observa?

Imprecisión

Debido a que los datos en punto flotante pueden representar más números pequeños que números grandes, también se tienen impresiones en grandes cifras.

```
cout.precision(20);

float datoFloat = 454545454871237;
cout<<datoFloat<<endl;
cout<<++datoFloat<<endl;

double datoDouble = 454545454871237;
cout<<datoDouble<<endl;
cout<<++datoDouble<<endl;
```

¿Que observa?

Imprecisión

Debido a que **double** contiene una mayor cantidad de bits (64) para la representación, se recomienda en lo posible usarlo en lugar de **float**.

Desbordamiento

Este problema ocurre cuando se intenta representar un valor numérico más grande que el tipo de dato puede manejar. Este problema se presenta en las operaciones que se realice. Pruebe:

```
int a = -2147483648;  
int b = 2147483647;  
cout << "Variable negativa a " << --a << endl;  
cout << "Variable positiva b " << ++b << endl;
```

¿Que observa?

Variables y tipos

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

Transformaciones de tipos

En algunas ocasiones algunas funciones retornan un tipo de dato dado y requerimos tratar la información en otro tipo **casting**. Pruebe:

```
int variable = 99;  
cout << variable << endl;  
cout << (char)variable << endl;
```

¿Que observa?

Transformaciones de tipos

Otro ejemplo:

```
double variable = 176.4;  
cout << variable << endl;  
cout << (int)variable << endl;
```

¿Que observa?

Texto

Las cadenas de texto se representan utilizando el tipo `char`. Este tipo de dato va entre -128 y 127, para un total de 128 posibles valores (entre 0 y 128).

Código ASCII

Existe un estándar internacional conocido como código ASCII, el cual asocia un número entre 0 y 255 a un carácter.

Variables y tipos

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

Código ASCII

TABLA DE CARACTERES DEL CÓDIGO ASCII

1	25	49	73	97	121	145	169	193	217	241
2	26	50	74	98	122	146	170	194	218	242
3	27	51	75	99	123	147	171	195	219	243
4	28	52	76	100	124	148	172	196	220	244
5	29	53	77	101	125	149	173	197	221	245
6	30	54	78	102	126	150	174	198	222	246
7	31	55	79	103	127	151	175	199	223	247
8	32	56	80	104	128	152	176	200	224	248
9	33	57	81	105	129	153	177	201	225	249
10	34	58	82	106	130	154	178	202	226	250
11	35	59	83	107	131	155	179	203	227	251
12	36	60	84	108	132	156	180	204	228	252
13	37	61	85	109	133	157	181	205	229	253
14	38	62	86	110	134	158	182	206	230	254
15	39	63	87	111	135	159	183	207	231	255
16	40	64	88	112	136	160	184	208	232	256
17	41	65	89	113	137	161	185	209	233	257
18	42	66	90	114	138	162	186	210	234	258
19	43	67	91	115	139	163	187	211	235	259
20	44	68	92	116	140	164	188	212	236	260
21	45	69	93	117	141	165	189	213	237	261
22	46	70	94	118	142	166	190	214	238	262
23	47	71	95	119	143	167	191	215	239	263
24	48	72	96	120	144	168	192	216	240	264

Transformaciones de tipos

Realice la siguiente prueba:

```
char letra = 113;  
cout << letra << endl;  
  
char otraLetra = 'f';  
cout << (int)otraLetra << endl;
```

¿Que observa?

Arreglos

Los arreglos en C++ son equivalentes a las listas que se vieron en fundamentos de programación. Existen dos tipos de arreglos:

- 1 **Arreglos estáticos:** Estos tienen un tamaño definido en el código.
- 2 **Arreglos dinámicos:** El tamaño de estos arreglos se define en ejecución. Los manejaremos a partir del momento que veamos Memoria Dinámica.

Arreglos

Para definir un arreglo estático se debe especificar:

```
<tipo> <nombre> [Tamaño];
```

Dimensiones

El arreglo que hemos visto anteriormente, es unidimensional, también si lo deseamos podemos definir más dimensiones

```
<tipo> <nombre> [Tamaño dimensión 1][Tamaño dimensión  
2]...[Tamaño dimensión n];
```

Arreglos

Pruebe el siguiente ejemplo

```
int cuadrados[100];

for(int i=0; i<100; i++){
    cuadrados[i] = i*i;
}

for(int i=0; i<100; i++){
    cout << cuadrados[i] << endl;
}
```

¿Que observa?. Recuerde que los elementos de un arreglo se indexan desde 0 hasta $n - 1$, donde n es el tamaño del arreglo.

Arreglos

Para conocer el tamaño de los arreglos podemos utilizar **sizeof** de la siguiente forma:

```
//sizeof(<arreglo>)/sizeof(<tipo>)
int cuadrados[100];
int size = sizeof(cuadrados)/sizeof(int);
for(int i=0; i<size; i++){
    cuadrados[i] = i*i;
}

bool arregloBool[200];
int sizeB = sizeof(arregloBool)/sizeof(bool);
for(int i=0; i<sizeB; i++){
    cout << arregloBool[i] << endl;
}
```

También observe. **Por defecto, ¡un arreglo se inicia con cualquier valor!**

Arreglos

Pruebe el siguiente ejemplo

```
int matriz[10][15];

for(int i=0; i<sizeof(matriz)/sizeof(matriz[0]); i++){
    for(int j=0; j<sizeof(matriz[0])/sizeof(int); j++){
        matriz[i][j] = i+j;
    }
}

for(int i=0; i<sizeof(matriz)/sizeof(matriz[0]); i++){
    for(int j=0; j<sizeof(matriz[0])/sizeof(int); j++){
        cout << matriz[i][j] << " ";
    }
    cout << endl;
}
```

Observe como se obtienen los tamaños de las filas y columnas.

Arreglos

Para arreglos n dimensionales

```
int matriz[10][15][19];
cout << "Primer arreglo" << endl;
cout << "Dimensión 1 " << sizeof(matriz)/sizeof(matriz[0])<< endl;
cout << "Dimensión 2 " << sizeof(matriz[0])/sizeof(matriz[0][0]) << endl;
cout << "Dimensión 3" << sizeof(matriz[0][0])/sizeof(int) << endl;

int matrizA[10][15][19][22][13];
cout << "Segundo arreglo" << endl;
cout << "Dimensión 1 " << sizeof(matrizA)/sizeof(matrizA[0])<< endl;
cout << "Dimensión 2" << sizeof(matrizA[0])/sizeof(matrizA[0][0]) << endl;
cout << "Dimensión 3" << sizeof(matrizA[0][0])/sizeof(matrizA[0][0][0]) <<
    endl;
cout << "Dimensión 4" << sizeof(matrizA[0][0][0])/sizeof(matrizA
    [0][0][0][0]) << endl;
cout << "Dimensión 5" << sizeof(matrizA[0][0][0][0])/sizeof(int) << endl;
```

Arreglos

Los arreglos estáticos se pueden definir directamente también así:

```
int primos [] = {1, 2, 3, 5, 7, 11, 13, 17, 23};  
cout << primos[2] << endl;
```

```
int matrizL[][4] = {{1, 2, 3, 4}, {2, 4, 6, 8}, {3, 6, 9, 12}};  
cout << matrizL[1][2] << endl;
```

```
int matriz3D[][4][4] = {{{1, 2, 3, 4}, {2, 4, 6, 8}, {3, 6, 9, 12}},  
                          {{1, 2, 3, 4}, {2, 4, 6, 8}, {3, 6, 9, 12}}};  
cout << matriz3D[1][2][1] << endl;
```

En el caso de los arreglos multidimensionales, es necesario especificar el tamaño de la dimensiones, excepto la primera.
¿Que observa?

Arreglos

Desarrolle programas:

- 1 Tiene definidos dos arreglos: que tiene los datos **(10,15,22,35,63)** y otro **(21,11,22,13,26)**. Debe retornar dos arreglos, el primero contiene la suma uno a uno de cada uno de los elementos del arreglo, el segundo sólo contiene los el elemento mayor comparado uno a uno.
- 2 Cree e imprima un arreglo bidimensional de 10 filas por 20 columnas. Este arreglo contiene en su primera fila los números desde 1 hasta 20, en la fila 2 son estos mismos pero cada uno multiplicado por 2 y así sucesivamente en la décima fila donde cada uno se multiplica por 10

Cadenas de texto

Las cadenas de texto son un caso especial de arreglos con el tipo **char**. En este caso sólo se declara el número de caracteres se cree va a tener el texto.

```
char texto[100]="Soy un pequeño pervertido";  
cout << texto << endl;
```

Pruebe este otro caso

```
char otroTexto[10]="Soy un pequeño pervertido";  
cout << otroTexto << endl;
```

¿Que observa?

Cadenas de texto

También en las cadenas de texto podemos incluir caracteres especiales para mejorar la presentación del texto, pruebe lo siguiente:

```
char texto[100]="Soy un pequeño\npervertido";  
cout << texto << endl;
```

Pruebe este otro caso

```
char otroTexto[100]="Soy un pequeño\tpervertido";  
cout << otroTexto << endl;
```

¿Que observa?

Variables especiales

Un caso especial de variables son las **constantes**, es decir variables que no pueden cambiar durante la ejecución. En C++ se utiliza la palabra **const** para declararlas

```
const double PI = 3.1416;  
double radio = 2;  
double area = PI*radio*radio;  
cout << area << endl;
```

¿Que observa?

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones**
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables
- 5 Strings
- 6 Contenedores STL
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios



Definición

Una función es una estructura que permite realizar una tarea específica y retornar un valor. Las funciones pueden tomar parámetros que modifiquen su funcionamiento. Las funciones son de gran utilidad en la programación ya que permiten descomponer grandes tareas complejas en tareas más pequeñas.

Definición

La estructura de las funciones en C++ es:

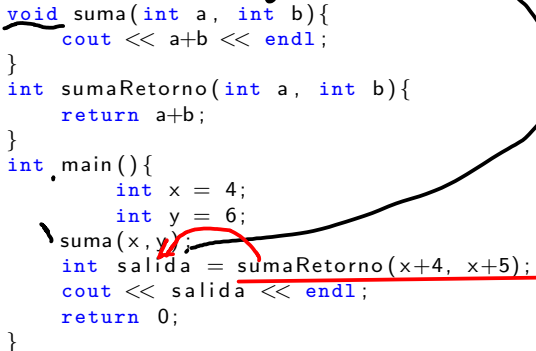
```
<tipo> [ clase :: ] <nombre> ( <parámetros> )  
{  
    cuerpo ;  
} return
```

[clase::] lo trataremos cuando veamos objetos :)

Definición

El tipo de una función es cualquier tipo válido para C++ y también puede ser un objeto. Existe un tipo especial llamado **void** que indica que no hay retorno. Construya el siguiente programa:

```
void suma(int a, int b){  
    cout << a+b << endl;  
}  
  
int sumaRetorno(int a, int b){  
    return a+b;  
}  
  
int main(){  
    int x = 4;  
    int y = 6;  
    suma(x, y);  
    int salida = sumaRetorno(x+4, x+5);  
    cout << salida << endl;  
    return 0;  
}
```



Ejercicio

Diseñe una calculadora, que funcione de la siguiente forma:

1 Solicite al usuario un número para indicar la operación:

1 1 para multiplicación

2 2 para división

3 3 para suma

4 4 para resta

2 Solicite dos números para realizar la operación

3 Muestre el resultado

Para esto implemente las funciones **suma**, **resta**,
multiplicación y **división**.

Ejercicio

Hay varias cosas que debemos solucionar:

- 1 Solicitar un número al usuario para saber que operación realizar
- 2 Después, solicitar dos números al usuario
- 3 Luego, realizar la operación
- 4 Finalmente, se debe validar con pruebas si su programa funciona

Definición

Construya el siguiente programa:

```
int main() {  
    cuadrado(22);  
}  
int cuadrado(int a) {  
    return a*a;  
}
```

¿Que observa?

Definición

Algunos lenguajes como C++ requieren que las funciones sean definidas antes de ser llamadas. Esto puede representar un problema ya que pueden existir llamados cruzados, por ejemplo función A llama a B y viceversa.

Solución


Cada lenguaje de programación tiene su propia solución a este problema, en nuestro lenguaje basta con definir la función (sin cuerpo) al inicio del programa.

Definición

Para solucionar el problema anterior, intente:

```
int cuadrado(int a);

int main(){
    cout << cuadrado(22) << endl;
    int entrada;
    cout << "Ingrese un número" << endl;
    cin >> entrada;
    cout << cuadrado(entrada) << endl;
}
int cuadrado(int a){
    return a*a;
}
```



¿Que observa?

Definición

Ahora, vamos a observar más de cerca como se envían parámetros a las funciones. Existen dos casos:

- 1 **Por valor:** Enviamos el valor de la variable en ese momento a la función, este es procesado y se emite una respuesta. **Es lo que hemos venido haciendo hasta ahora**
- 2 **Por referencia:** Enviamos una referencia de la variable (su localización) y se procesa. **Es lo nuevo :)**

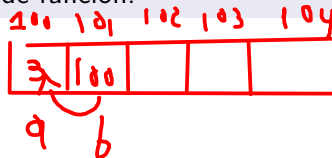


Handwritten notes illustrating parameter passing by reference. The text shows `funcion(q)` and `funcion(&x)`, with a red arrow pointing from `&x` to `funcion(&x)`, indicating that the address of `x` is passed to the function.

Referencia

Una referencia es la ubicación de una variable en la memoria. Esto nos va servir para trabajar únicamente con la ubicación de la variable y no su valor, lo que nos permite utilizar la memoria de forma más eficiente, al evitar estar creando variables cada vez que se hace un llamado de función.

```
int a = 3
int b = &a
```



Definición

En C++ se utiliza el operador **&** antes del nombre de la variable para hacer paso por referencia, observe:

```
int funcion(int &a, int &b){  
    //Lo que hace el código  
}
```

Paso de parámetros

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición

Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector

Mapas

Pilas y colas

Ejercicios

Definición

Pruebe el siguiente código

```
int variable = 3;  
cout << variable << endl;  
cout << &variable << endl;
```

¿Que observa?

Paso de parámetros

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

Definición

Pruebe el siguiente código de paso de parámetros **por valor**

```
int funcionVal(int a, int b){  
    a = 2*(b+a);  
    b = 2*(b-a);  
    return a*b;  
}  
  
int main(){  
    int entradaA;  
    int entradaB;  
    cin >> entradaA;  
    cin >> entradaB;  
    cout << funcionVal(entradaA, entradaB) << endl;  
    cout << entradaA << endl;  
    cout << entradaB << endl;  
}
```

¿Que observa?

Paso de parámetros

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector

Mapas

Pilas y colas

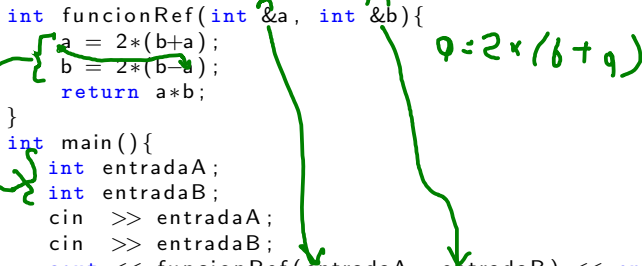
Ejercicios

Definición

Pruebe el siguiente código de paso de parámetros **por referencia**

```
int funcionRef(int &a, int &b){  
    a = 2*(b+a);  
    b = 2*(b-a);  
    return a*b;  
}  
  
int main(){  
    int entradaA;  
    int entradaB;  
    cin >> entradaA;  
    cin >> entradaB;  
    cout << funcionRef(entradaA, entradaB) << endl;  
    cout << entradaA << endl;  
    cout << entradaB << endl;  
}
```

$q = 2 * (b + a)$



¿Que observa?

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables**
- 5 Strings
- 6 Contenedores STL
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios

Definición

Las reglas de alcance de las variables indican en que lugar de su programa una variable es válida, cuando es construida y destruida. Este es un paso elemental en el manejo de memoria de los programas. De acuerdo al alcance de variables, este puede ser estático o dinámico (dependiendo del lenguaje)

- 1 Dinámico:** Lo defino el lenguaje de programación, es eficiente pero complejo de manejar. Ya que este se determina en ejecución.
- 2 Estático:** Es el que vamos a trabajar, el programador lo define.

Alcance de las variables

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

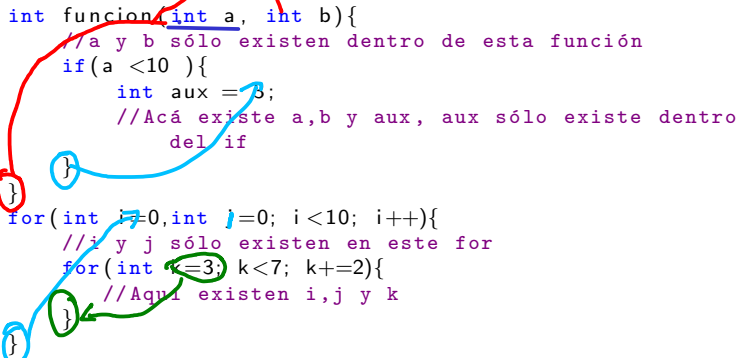
Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

Alcance

El alcance de las variables está limitado a la estructura donde se ha definido:



```
int funcion(int a, int b){  
    //a y b sólo existen dentro de esta función  
    if(a <10 ){  
        int aux = 3;  
        //Acá existe a,b y aux, aux sólo existe dentro  
        del if  
    }  
    for(int i=0,int j=0; i<10; i++){  
        //i y j sólo existen en este for  
        for(int k=3; k<7; k+=2){  
            //Aquí existen i,j y k  
        }  
    }  
}
```

The diagram uses handwritten annotations to show variable scope:

 - A red circle highlights the opening brace of the `funcion` function.
 - A red arrow points from this circle to the parameter `int a`.
 - A blue circle highlights the closing brace of the `if` statement.
 - A blue arrow points from this circle to the variable `aux` inside the `if` block.
 - A green circle highlights the opening brace of the innermost `for` loop.
 - A green arrow points from this circle to the variable `k` inside the loop.
 - A blue circle highlights the closing brace of the outermost `for` loop.
 - A blue arrow points from this circle to the variable `i` in the loop header.

Alcance

En el alcance estático se manejan dos tipos de alcance:

- 1 **Alcance local:** Es el que estamos manejando, las variables sólo son visibles dentro de sus estructuras, es el que hemos venido manejando
- 2 **Alcance global:** La variable es visible en todo el programa.

Alcance de las variables

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

Alcance global

Para que una variable sea visible en todo el programa, solamente basta con declararla en el inicio del mismo:

```
#include <iostream>

using namespace std;

int variableA = 0;
int variableB = 0;
//...
```

Alcance global

Sin embargo, **esta es una pésima práctica de programación**.
Trate de no programar con variables globales debido a:

- 1 Cualquiera puede modificar la variable ¿Y si su código es muy extenso? ¿Y si modifica sin querer su variable global pensando que es otra?
- 2 Las funciones pueden recibir y retornar datos. Si necesita modificar una variable de una función a otra **use paso por referencia**

Alcance de las variables

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios



Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables
- 5 Strings**
- 6 Contenedores STL
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios

Definición

Este librería permite trabajar con cadenas de texto debido a las dificultades que representa trabajar con char. Para trabajar con ella es suficiente:

```
#include <string>
```

Definición

Esta librería permite declarar, por ejemplo arreglos de cadenas de texto:

```
string nombres[] = { "Juan", "Pedro", "Marcos" };
```


Definición

También permite realizar operaciones:

```
string nombre = "Carlos";  
//En el caso de la comparación, se utiliza .compare, este  
    retorna 0 si son iguales  
//Por ende debe usar un operador relacional == 0 para  
    validar  
bool comparacion = (nombre.compare("Juan") == 0);  
cout << comparacion << endl;  
//Permite concatenar palabras  
string textoA = "gato";  
string textoB = " es rojo";  
string textoC;  
textoC.append(textoA);  
textoC.append(textoB);  
cout << textoC << endl;
```

Definición

También permite realizar operaciones:

```
//Para recorrer un string
string recorrer = "Soy un excelente string";
for(int i=0; i<(int)recorrer.length(); i++){
    cout<<recorrer[i]<<endl;
}
//
```

Para explorar más funciones consulte

<http://www.cplusplus.com/reference/string/string/>

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables
- 5 Strings
- 6 Contenedores STL**
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios

Definición

Vector es un contenedor de cualquier tipo, es decir, puede contener cadenas de texto, booleanos, enteros, objetos, etc. El vector puede verse como un array, con la diferencia que puede contener cualquier número de elementos.

```
#include <vector>
```

Definición

Vector es un contenedor de cualquier tipo, es decir, puede contener cadenas de texto, booleanos, enteros, objetos, etc. El vector puede verse como un arreglo, con la diferencia que puede contener cualquier número de elementos. Para utilizar vector incluya:

```
#include <vector>
```

hinc luda <vector>

Definición

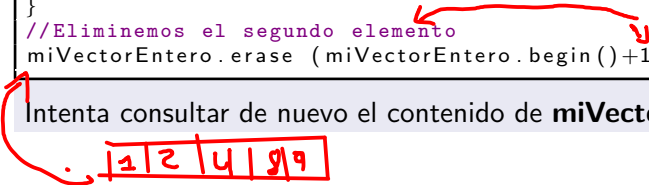
Este nos permite a través de algunas funciones: insertar, consultar o eliminar elementos.

```
//Creamos un vector de enteros
vector<int> miVectorEntero;
//Creamos un vector de cadenas de texto
vector<string> miVectorTexto;
//Insertamos
miVectorEntero.push_back(999); //Posición 0 del vector
miVectorEntero.push_back(888); //Posición 1 del vector
miVectorTexto.push_back("Hola"); //Posición 0 del vector
miVectorTexto.push_back("Como estás"); //Posición 1 del
vector
```

Definición

Este nos permite a través de algunas funciones: insertar, consultar o eliminar elementos.

```
//Consultemos los datos
for(int i=0; i<miVectorEntero.size(); i++){
    cout << miVectorEntero[i] << endl;
}
//Eliminemos el segundo elemento
miVectorEntero.erase(miVectorEntero.begin()+1);
```



Intenta consultar de nuevo el contenido de **miVectorEntero**

. [1 | 2 | 4 | 9 | 9]

[1 | 4 | 8 | 9]

Ejercicio

Utilizando vectores

- 1 Genere una aplicación usando vectores que permita almacenar la cédula (numérico) y el nombre de una persona (string)
- 2 Genere una aplicación usando vectores que permita almacenar un código alfanumérico (string) y un valor de punto flotante (double)

Las aplicaciones deben primero solicitar el número de datos que se van a ingresar, solicitarlos y finalmente imprimirlos en pantalla.

Definición

Los mapas son contenedores asociativos para contener en orden una lista de parejas de valores únicos asociados como clave/valor. Para utilizarlos debe incluir:

```
#include <map>
```

Definición

Para crear un mapa debemos indicar el par de clave y valor.

```
map<char, int> codigos;  
codigos['a'] = 1000;
```

Definición

Para recorrer un mapa requerimos iteradores:

```
for (map<char, int>::iterator it=codigos.begin(); it!=  
    codigos.end(); ++it){  
    cout << it->first << " => " << it->second << '\n'<<  
        endl;  
}
```

Un iterador es un objeto que se mueve a través de un contenedor de otros objetos y selecciona a uno de ellos cada vez

Definición

Podemos eliminar elementos especificando su llave.

```
codigos.erase('c');
```

Hay varias formas de realizarlo utilizando iteradores, para más detalles consulte el API.

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}
 \begin{array}{|c|c|} \hline 5 & 8 \\ \hline 9 & 10 \\ \hline \end{array}
 =
 \begin{array}{|c|c|} \hline 5+18 & 8+20 \\ \hline 15+36 & 24+40 \\ \hline \end{array}$$

$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$, 2 2, $\begin{array}{|c|} \hline 2 \\ \hline \end{array}$

$$\begin{array}{|c|c|} \hline 23 & 28 \\ \hline 51 & 64 \\ \hline \end{array}$$

Ejercicio

Utilizando mapas

- 1 Genere una aplicación usando mapas que permita almacenar la cédula (numérico) y el nombre de una persona (string)
- 2 Genere una aplicación usando mapas que permita almacenar un código alfanumérico (string) y un valor de punto flotante (double)

Las aplicaciones deben primero solicitar el número de datos que se van a ingresar, solicitarlos y finalmente imprimirlos en pantalla. ¿Que diferencia observa con vectores?

Definición

Las pilas son estructuras de datos que tienen dos operaciones básicas: **push** (para insertar un elemento) y **pop** (para extraer un elemento). Su característica fundamental es que al extraer se obtiene siempre el último elemento que acaba de insertarse. Por esta razón también se conocen como estructuras de datos **LIFO** (del inglés **Last In First Out**). Para utilizarlas:

```
#include <stack>
```

Definición

Las pilas se utilizan en muchas aplicaciones que utilizamos con frecuencia. Por ejemplo, la gestión de ventanas en Windows (cuando cerramos una ventana siempre recuperamos la que teníamos detrás). Otro ejemplo es la evaluación general de cualquier expresión matemática para evitar tener que calcular el número de variables temporales que hacen falta

$$3 + 4 * (8 - 2 * 5)$$

5
-2
8
4
3

-10
8
4
3

-2
4
3

-8
3

-5

Definición

Para crear una pila debemos indicar que tipo de dato se va almacenar en ella.

```
stack<int> pilaNumeros;
```

Definición

Podemos realizar diferentes operaciones

```
//Insertar un elemento  
pilaNumeros.push(2);  
//Eliminar el elemento que está en la cima de la pila  
pilaNumeros.pop();  
//Conocer el elemento que está en la cima de la pila  
pilaNumeros.top()
```

Ejercicio

Utilizando pilas, genere las operaciones de:

1 $2 + 3 * 5 + 8 * 6$

2
$$\frac{10+5*(6+8)+5}{\sqrt{10+5*6+\frac{8}{2}}}$$

Pista: Utilice dos pilas, una para almacenar el número y otra la operación. Recuerde la prioridad de los operadores.

Definición

Las colas también son llamadas FIFO (First In First Out), que quiere decir “el primero que entra es el primero que sale”.

```
#include <queue>
```

Aplicaciones

Las colas son muy útiles en campos como la simulación (para simular colas), el análisis de algoritmos (como estructura para almacenar nodos de un árbol en un recorrido específico), entre otros.

Definición

Para crear una cola debemos indicar que tipo de dato se va almacenar en ella.

```
queue<int> colaNumeros;
```

Definición

Podemos realizar diferentes operaciones en las colas

```
//Insertar  
colaNumeros.push(5);  
//Consultar el primero de la cola  
colaNumeros.front()  
//Eliminar el primero de la cola  
colaNumeros.pop()
```

Ejercicio

Utilizando colas, genere la simulación de una fila de un banco.

- 1 La aplicación tiene 3 opciones: 1 para ingresar una persona a la cola, 2 para atender y 3 para salir.
- 2 Si se selecciona la opción 1, se pide el nombre de la persona. Posteriormente muestra la opciones y queda a la espera se ingrese otra opción.
- 3 Si se selecciona la opción 2, se atiende a la persona. Se imprime en pantalla un mensaje indicando que la persona x ha sido atendida
- 4 Si se selecciona la opción 3, se termina la aplicación.

Pista: Utilice un **do-while** para crear el menú de la aplicación.

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

- 1 Expresiones
- 2 Variables y tipos
- 3 Funciones
 - Definición
 - Paso de parámetros
- 4 Alcance de las variables
- 5 Strings
- 6 Contenedores STL
 - Vector
 - Mapas
 - Pilas y colas
- 7 Ejercicios**

Ejercicios

Muchachos, ahora somos capaces de hacer cosas un tanto más complejas :): Asuman estos retos:

- 1 Diseñar un programa, que recolecte las edades y nombres de 10 personas. Usando arreglos imprima para cada persona el mensaje "<Nombre> tiene <Edad> años". Para los datos de texto utilice **string**
- 2 Haga una aplicación para una tienda de mascotas que:
 - 1 Tenga almacenado el código (numérico) y el precio de 10 productos (use dos arreglos)
 - 2 Pida al usuario el código y la cantidad de ese producto que desea comprar
 - 3 Si el producto existe, le indica al usuario el dinero que debe cancelar 'item Permita eliminar elementos

Ejercicios

Muchachos, ahora somos capaces de hacer cosas un tanto más complejas :): Asume este reto:

Diseñar un programa, que tenga almacenados los datos de cédula, nombre, nombre EPS y nombre banco de un grupo de personas (utilice 4 mapas para esto). Esta aplicación permite

- Buscar una persona por su cédula, e imprimir la información
- Imprimir la información de todas las personas
- Eliminar la información de una persona por cédula

¿Preguntas?

Introducción a
la
programación
orientada a
objetos
(IPOO)

Carlos Andrés
Delgado S.

Expresiones

Variables y
tipos

Funciones

Definición
Paso de
parámetros

Alcance de las
variables

Strings

Contenedores
STL

Vector
Mapas
Pilas y colas

Ejercicios

