REPASO
FUNCIONES

» Ventre, Luis O.

Funciones REPASO

Una FUNCION:

- Que es? ….
- Para que sirve? …
- Como hago para declararla?
- Como hago para escribirla?
- Como hago para utilizarla?
- Como le envío valores para que la misma resuelva su objetivo?

Funciones REPASO

Una FUNCION:

- Que es? ….
- Unidad de código, que me permite DIVIDIR la complejidad de un problema en BLOQUES.

- Para que sirve? …
- Para simplificar la codificación, testing, mantenimiento, y reutilización del código.

- Como hago para declarar una?
- Se debe escribir una línea de código llamada PROTOTIPO de la función.

```
tipo-de-datos-a-devolver  nombre-de-función  (lista de tipos de datos argumento);
```

Funciones REPASO

Una FUNCION:

- Como hago para escribir una?
- Es necesario, un ENCABEZADO de función, y su cuerpo. Este código se escribe luego de la función MAIN.

- Como hago para utilizarla?
- Para utilizar una función, solo debo INVOCARLA; o colocar su nombre en mi programa, y entre paréntesis enviarle los datos necesario para su correcto funcionamiento.

- Como le envío argumentos para que resuelva su objetivo?
- Existen dos maneras. **Por valor** y **por referencia**.

Funciones REPASO

- **Pasaje de argumentos por VALOR:**

- Al llamar la función, como argumento envió nombre de las variables. Esto genera una copia del valor de las mismas en las variables que declare en el ENCABEZADO de la función.

- De esta forma, la función NO PUEDE MODIFICAR los valores de las variables de la función MAIN.

- La función DISPONE de una COPIA de los valores de las variables enviadas.

Devolver un solo valor

```
#include <iostream>
using namespace std;

int encontrarMax(int, int);  // prototipo de la funcion

int main()
{
  int primernum, segundonum, max;

  cout << "\nIngrese un numero: ";
  cin  >> primernum;
  cout << "Bien! Por favor ingrese un segundo numero: ";
  cin  >> segundonum;

  // la funcion es llamada aqui
  max = encontrarMax(primernum, segundonum);

  cout << "\nEl maximo de los dos numeros es: " << max << endl;

  system("PAUSE");
  return 0;
}
```

Alerta a main y demás fx sobre el valor devuelto

Asignación a la variable max del valor devuelto por la función

Devolver un solo valor

```
int encontrarMax(int x, int y)
{                       // inicio del cuerpo de la función
  int maxnum;           // declaracion de variable

  if (x >= y)           // encuentra el numero máximo
    maxnum = x;
  else
    maxnum = y;

  return maxnum;        // instrucción return
}
```

Encabezado declara que valor será devuelto

Ver en el libro ejemplo similar programa 6.6

- Unidad 4

FUNCIONES continuación

(Capitulo 6 bibliografía)

» Ventre, Luis O.

Devolver múltiples valores

Existen ocasiones en donde es necesario darle a la función llamada acceso directo a las variables de la función que llama.

Para lograr esto se necesita enviarle a la función llamada, **la dirección de la variable.**

Una vez que la función llamada conoce la dirección de la variable "conoce donde vive" y puede tener acceso y cambiar su valor de manera directa.

Este tipo de transmisión se llama **"pasaje o transmisión por referencia".**

C++ proporciona dos tipos de parámetros de dirección, referencia y apuntadores.

Variables de referencia

Una vez que se ha declarado una variable se le pueden asignar nombres adicionales o alias; esto se logra usando una declaración de referencia**:**

tipo-de-datos& nuevo-nombre = nombre-existente

int& suma = total;

Esta declaración de referencia iguala el nombre **suma al nombre total** y ambos refieren a la **misma variable**. La modificación de cualquiera de ellos afectara el valor de la variable.

- Se debe tener especial cuidad que la referencia sea <u>del mismo tipo de datos</u> que la variable a la que se refiere.
- No igualar una referencia a un numero.

```
int main()
{
    int b,c;
    int& a=b;
    c=b=0;
    a=c;
    a=1;
    cout<<c;
    system("pause");
}
```

Transmisión de parámetros de referencia

La invocación a una función con pasaje por referencia desde el emisor es idéntica. Para que se transmita la dirección **debe declararse los tipos de parámetros como referencia**; esto se hace con la siguiente sintaxis:

```
tipo-de-datos& nombre-referencia
```

Ej.:

```
double& num1;
```

Indica que num1 es un parámetro de referencia que se utilizara para almacenar la dirección de un double. O leer al revés por ejemplo num1 es la dirección de una variable de precisión doble.

Transmisión de parámetros de referencia

A continuación veremos un ejemplo donde desde una función se modifican los valores originales de la variables de main.

Como es necesario modificar dos valores, como parámetros o argumentos se necesitaran **dos referencias a esos valores**. Como la función solo modifica los valores y no devuelve nada en su encabezado vemos el inicio con la **palabra "void".**

Un prototipo adecuado para una función así descripta seria:

```
void valnuevo(double&, double&);
```

Transmisión de parámetros de referencia

```
#include <iostream>
using namespace std;

void valornuevo(double&, double&);  // prototipo con dos parámetros
                                    // referencia (&)
int main()
{
  double primernum, segundonum;

  cout << "Ingrese dos numeros: ";
  cin  >> primernum >> segundonum;
  cout << "\nEl valor del primer numero es: " << primernum << endl;
  cout << "El valor del segundo numero es: "  << segundonum << "\n\n";

  valornuevo(primernum, segundonum);   // llamado de la función

  cout << "Ahora el valor del primer numero es: "  << primernum << endl;
  cout << "Ahora el valor del segundo numero es: " << segundonum << endl;

  system("PAUSE");
  return 0;
}
```

Transmisión de parámetros de referencia

```
void valornuevo(double& xnum, double& ynum)
{
    cout << "El  valor de xnum es: " << xnum << endl;
    cout << "El  valor de ynum es: " << ynum << "\n\n";
    xnum = 89.5;
    ynum = 99.5;

   return;
}
```

```
Ingrese dos numeros: 15
20

El valor del primer numero es: 15
El valor del segundo numero es: 20

El  valor de xnum es: 15
El  valor de ynum es: 20

Ahora el valor del primer numero es: 89.5
Ahora el valor del segundo numero es: 99.5
Presione una tecla para continuar . . .
```

Transmisión de parámetros de referencia

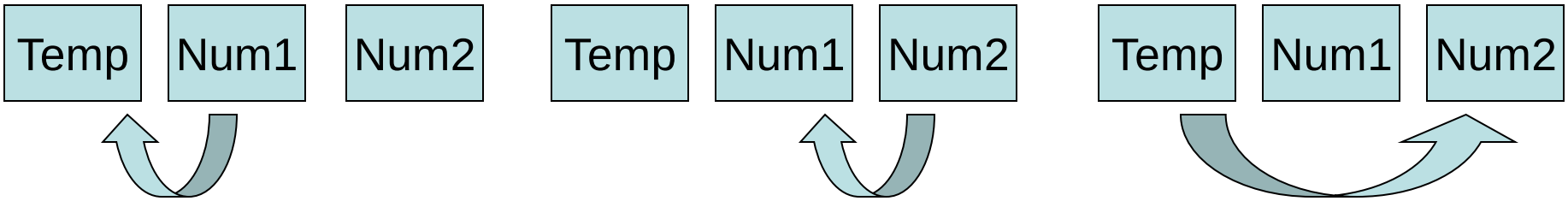
Un ejemplo útil es el algoritmo de una función que de alguna manera me intercambie los valores de dos variables.

La forma de hacer esto es:

1-Guarda el valor del primer parámetro en una ubicación temporal.

2-Almacenar el valor del segundo parámetro en la primera variable.

3-Almacenar el valor temporal en el segundo parámetro



Como debe afectarse mas de una variable, y deben modificarse directamente las variables del main, este ejemplo es un caso típico a resolverse con pasaje por referencia

Transmisión de parámetros de referencia

```
#include <iostream>
using namespace std;

// la funcion recibe 2 referencias
void intercambio(double&, double&);

int main()
{
  double primernum = 20.5, segundonum = 6.25;

  cout << "El valor almacenado en primernum es: "  << primernum << endl;
  cout << "El valor almacenado en segundonum es: " << segundonum << "\n\n";

  // llamado a la función con referencias
  intercambio(primernum, segundonum);

  cout << "Ahora el valor almacenado en primernum es: "
       << primernum << endl;
  cout << "Ahora el valor almacenado en segundonum es: "
       << segundonum << endl;

  system("PAUSE");
  return 0;
}
```

Transmisión de parámetros de referencia

```
void intercambio(double& num1, double& num2)
{
  double temp;

  temp = num1;      // guarda el valor de num1
  num1 = num2;      // almacena el valor de num2 en num1
  num2 = temp;      // cambia el valor de num2

  return;
}
```

Aquí se cambian los valores de primernum y segundonum. Por valor es imposible

```
El valor almacenado en primernum es: 20.5
El valor almacenado en segundonum es: 6.25

Ahora el valor almacenado en primernum es: 6.25
Ahora el valor almacenado en segundonum es: 20.5
Presione una tecla para continuar . . .
```

Funciones IN LINE

La invocación a una función, impone cierta sobrecarga a la computadora, guardar los valores de los argumentos en el stack o pila, espacio reservado de memoria, transmitir el control a la función, cargar el valor devuelto en la pila y devolver el control a la función principal.

Esta sobrecarga se justifica cuando la función es importante y llamada muchas veces en el programa. Para ahorrar líneas de código y simplificar el programa.

Ventaja: Aumento en la velocidad de ejecución

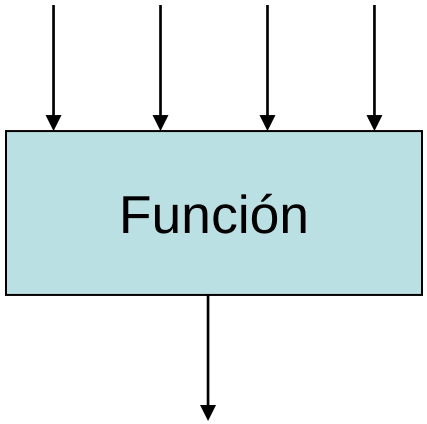
Con funciones simples, es mas fácil colocarlas bajo un nombre y que el compilador las reemplace en el código cuando encuentre su invocación. Esto se logra con la palabra reservada "inline"

Desventaja: Aumento en el tamaño del programa

Indicarle a un compilador que la función es inline produce una copia directa del código de la función en el lugar donde es invocada.

Alcance de una variable

Con programas con mas de una función, podemos observar las variables descriptas dentro de cada función. Estas funciones pueden ser vistas como cajas negras independientes.



ALCANCE – Sección del programa donde el identificador, como una variable, es valido o "conocido".

El alcance de una variable puede ser LOCAL o GLOBAL.

Una variable creada dentro de una función esta disponible solo para la función en si, **son variables locales a la función.**

Alcance de una variable

Ya que el uso de una variable es relevante cuando se usa en expresiones dentro de la función donde se declaro, pueden existir numerosas variables con el mismo nombre, declaradas en cada función, para cada función se crea una **variable separada y distinta.**

Al colocar la instrucción de declaración dentro de una función define la variable como local.

Como veremos las variables pueden declararse fuera de las funciones, ***estas variables tienen un alcance global****. Y pueden ser utilizadas por todas las funciones que se coloquen en el programa después de su declaración.*

Observación: el alcance de una variable no afecta a su tipo. Ver página 364 libro.

Alcance de una variable

```
#include <iostream>
using namespace std;

int primernum;      // crea una variable global llamada primernum

void valfun();      // prototipo de la función (declaracion)

int main()
{
  int segundonum;   // crea una variable local llamada segundonum

  primernum  = 10;  // almacena un valor en la variable global
  segundonum = 20;  // almacena un valor en la variable local

  cout << "De main(): primernum  = "  << primernum  << endl;
  cout << "De main(): segundonum = " << segundonum << endl;

  valfun();         // llama a la funcion valfun

  cout << "\nDe main() de nuevo: primernum  = " << primernum  << endl;
  cout <<   "De main() de nuevo: segundonum = " << segundonum << endl;

  system("PAUSE");
  return 0;
}
```

Alcance de una variable

```
void valfun()       // no se pasan valores a esta función
{
  int segundonum;  // crea una segunada variable local llamada segundonum

  segundonum = 30; // esto sólo afecta al valor de la variable local

  cout << "\nDe valfun(): primernum  = " << primernum  << endl;
  cout <<   "De valfun(): segundonum = " << segundonum << endl;

  primernum = 40;    // esto cambia primernum en ambas funciones

  return;
}
```

```
De main(): primernum  = 10
De main(): segundonum = 20

De valfun(): primernum  = 10
De valfun(): segundonum = 30

De main() de nuevo: primernum  = 40
De main() de nuevo: segundonum = 20
Presione una tecla para continuar . . .
```

Alcance de una variable

OPERADOR DE RESOLUCION DE ALCANCE

Cuando una variable local tiene el mismo nombre que una variable global, todas las veces que se haga referencia a su nombre dentro de su alcance local se refiere a esta.

Para desde el alcance local acceder a una variable de igual nombre pero global, debe utilizarse el **operador de resolución de alcance "::"**

Cuando se usa de esta manera le indica al compilador que utilice la variable global.

```
#include<iostream>
using namespace std;

double numero=42.5;

int main()
{
    double numero =26.4;
    cout<<" El valor de numero es "<< numero<<endl;
    cout<<" El valor de numero es "<< ::numero<<endl;
```

```
 El valor de numero es 26.4
 El valor de numero es 42.5
Presione una tecla para continuar . . .
```

MAL USO DE VARIABLES GLOBALES

Las variables globales permiten al programador **“saltarse”** las salvaguardas normales proporcionadas por las funciones. En lugar de transmitir variables a una función es posible hacer todas las variables globales – **NO HAGA ESTO.**

Las funciones dejarían de ser independientes y aisladas entre si. Se vuelve imposible seguir el hilo de un programa al depurarlo, y lograr determinar un error de asignación de un valor puede ser un caos.

Sin embargo, es lógico encontrar programas grandes, con un numero acotado de variables y constantes globales que pueden ser accedidas desde las funciones sin tener que pasárseles a todas como parámetros.

Esto no se extiende a los prototipos de funciones. Todos los vistos son globales. Un prototipo dentro de una fx hace una declaración local.

Unidad 5

Arreglos REPASO

y continuación

(Capitulo 11 bibliografía)

» Ventre, Luis O.

Repaso ARREGLOS

ARREGLOS:

- Que es-son? ….
- Para que sirve? …
- Como hago para declararlo?
- Como hago para utilizarla?
- Como cargar o enviar todos los valores de un arreglo?
- Arreglos unidimensionales, bidimensionales?
- Arreglos como argumentos de funciones?

Repaso ARREGLOS

ARREGLOS:

- Que es-son?
- Lista definida de elementos de un mismo tipo, generadas como grupo bajo un mismo nombre.

- Para que sirve?
- Simplificar el agrupamiento, la declaración, y el uso de un grupo de elementos del mismo tipo.

- Como hago para declararlo?
- Debe declararse de manera similar con la declaración de una variable única, salvando la diferencia de colocar la cantidad de elementos del grupo entre corchetes.

```
tipodedato nombre[cantidadelementos]
```

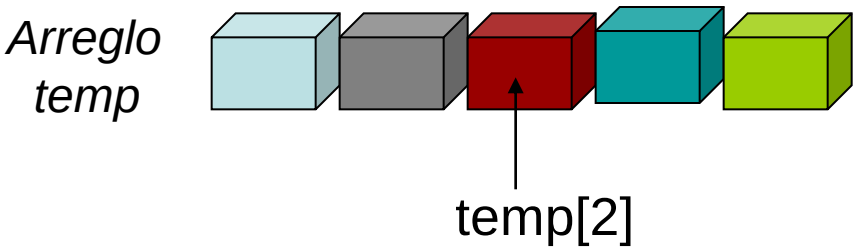
```
int  arreglo[8]
```

Repaso ARREGLOS

ARREGLOS:

- Como hago para utilizarlo?
- Puedo referirme a los elementos del arreglo indicando el nombre del mismo y el subíndice

```
arreglo[5]=0;
```



- Como cargar o enviar todos los valores de un arreglo?
- Para ello, se utilizan instrucciones de bucle como la instrucción FOR, en la cual la variable contadora también es la que indica el subíndice del elemento a utilizar.

```
for (i=0;i<5;i++)
    arreglo1[i]=0;
```

Repaso ARREGLOS

ARREGLOS:

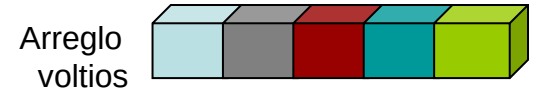
- Unidimensionales?
- Son grupos de elemento de un mismo tipo, que pueden inicializarse, recorrerse y utilizarse a través de una sola instrucción FOR.

- Bidimensionales?
- Arreglos, formados por filas y columnas, llamados también matrices, las cuales para inicializar, y utilizar son necesarios dos bucles FOR anidados, uno recorriendo sus filas y otro interior recorriendo las columnas.

- Arreglos como argumentos de funciones? Valor y referencia.

RECORDAR QUE TODO ARREGLO TIENE SU PRIMER ELEMENTO CON SUBINDICE 0.

Arreglos Como ARGUMENTOS



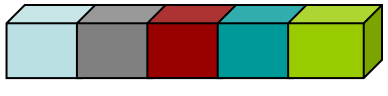
- Los elementos de un arreglo, **puede enviarse a una FUNCION**, como **argumentos**, de la misma forma que una variable. Simplemente colocando el nombre del arreglo y el subíndice del componente a enviar:

```
hallarmin(voltios[2], voltios[5]);
```

- Cuando se transmite de esta manera un elemento de un arreglo a una función, en la misma se crea una variable con una **COPIA del valor original del arreglo**, tal cual ocurría en el **“pasaje por valor”** de argumentos.

- En casos donde nuestro **arreglo es grande**, pasar un arreglo completo de esta forma generaría una copia completa de arreglo para cada función y seria un **desperdicio de recursos/memoria**.

Arreglos Como ARGUMENTOS



- Ante este problema es posible poner a disposición de la función todo el arreglo. Esto se logra con la siguiente llamada:

```
hallarmin(voltios);
```

- Esta llamada, envía a la función **la dirección del primer componente del arreglo**, y de esta manera en la función se accede directamente a todo el arreglo de manera análoga a como sucedía con el **"pasaje por referencia".**

- En el caso del llamado anterior la función llamada debe ser **alertada** que el argumento es un arreglo por lo que su encabezado será:

```
int hallarmin (int voltios[5])
{….
```

Arreglos Como ARGUMENTOS

```
#include <iostream>
using namespace std;

const int MAXELS = 5;
int encontrarMax(int [MAXELS]);     // prototipo de la función

int main()
{
  int nums[MAXELS] = {2, 18, 1, 27, 16};

  cout << "El valor maximo es " << encontrarMax(nums) << endl;

  system("PAUSE");
  return 0;
}

// encontrar el valor maximo
int encontrarMax(int vals[MAXELS])
{
  int i, max = vals[0];

  for (i = 1; i < MAXELS; i++)
    if (max < vals[i]) max = vals[i];

  return max;
}
```

Devuelve un entero
Recibe como argumento un arreglo de 5 elementos

Llamado a función sin cambios

```
El valor maximo es 27
Presione una tecla para continuar . . .
```

En este programa se crea un solo arreglo, se conoce desde main como nums y desde la función como vals.

Arreglos Como ARGUMENTOS

En el encabezado de la función anterior, ya que se le transmite la dirección del elemento 0 del arreglo, **no es imprescindible** colocar el **tamaño del arreglo**. Incluso es mas entendible omitirlo; por lo que el encabezado de la función recibiría el arreglo y el numero de elementos como argumento:

```
int encontrarMax(int [], int);     // prototipo de la función
.
.
.


int encontrarMax(int vals[], int numels)
{
  int i, max = vals[0];

  for (i = 1; i < numels; i++)
    if (max < vals[i]) max = vals[i];

  return max;
}
```

Arreglos Como ARGUMENTOS

De manera idéntica se procede para el envío de un arreglo BIDIMENSIONAL como argumento de una función; el llamado es igual, y la recepción en la función llamada solo deberá especificar los subíndices como se muestra:

Para las declaraciones de arreglos como:

```
int prueba[3][4];
float factores[26][10];
```

las siguientes llamadas a función son validas:

```
hallarMax(prueba);
obtener(factores);
```

```
#include <iostream>
#include <iomanip>
using namespace std;

const int RENGLONES = 3;
const int COLUMNAS  = 4;

void despliega(int [RENGLONES][COLUMNAS]); // prototipo de la función

int main()
{

  int val[RENGLONES][COLUMNAS] = {8,16,9,52,
                                  3,15,27,6,
                                  14,25,2,10};

  despliega(val);

  system("PAUSE");
  re
}
```

```
   8  16   9  52
   3  15  27   6
  14  25   2  10
Presione una tecla para continuar . . . _
```

```
void despliega(int nums[RENGLONES][COLUMNAS])
{
  int num_renglon, num_colum;
  for (num_renglon = 0; num_renglon < RENGLONES; num_renglon++)
  {
    for(num_colum = 0; num_colum < COLUMNAS; num_colum++)
      cout << setw(4) <<nums[num_renglon][num_colum];
    cout << endl;
  }

  return;
}
```

Llamado a función

Alerto a la función que estará disponible un arreglo bidimensional

En este programa se crea un solo arreglo bidimensional que se conoce desde main como val y desde la función como nums

Arreglos Como ARGUMENTOS

De la misma manera que en los arreglos unidimensionales se puede **omitir el tamaño** en los arreglos bidimensionales puede omitirse el **numero de filas**; pero no el de columnas. Esto naturalmente se observa por la forma de almacenar datos del compilador en el arreglo; en donde el numero de columnas es necesario para el calculo del desplazamiento. Ver detalle pagina 642 bibliografía.

Podría encontrarse una declaración correcta:

desplegar (int nums [][4]);