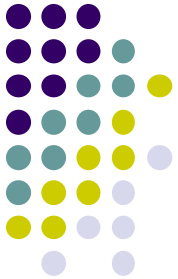
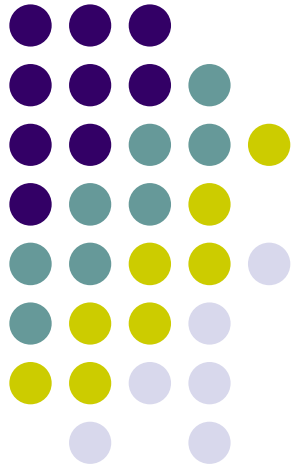
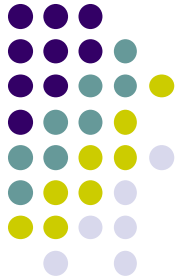


Modularización

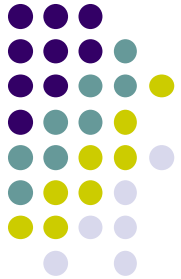


Programación en Bloque Monolítico



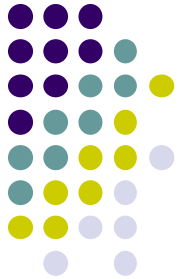
- Pocas variables representando diferentes cosas;
- Inviabiliza grandes sistemas de programación
  - Un único programador pues no hay división del programa
  - Inducción a errores por causa de la visibilidad de variables y flujo de control irrestricto
  - Dificulta la reutilización de código
- Eficiencia de programación pasa a ser cuello de botella

Proceso de Resolución de Problemas Complejos



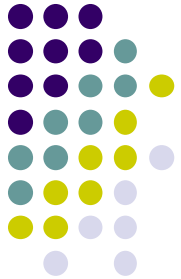
- Uso de Dividir para Conquistar
  - Resolución de varios problemas menos complejos
  - Aumenta las posibilidades de reutilización
- Técnicas de Modularización objetivan Dividir para Conquistar
  - Tornan más fácil el entendimiento del programa
  - Segmentan el programa
  - Encapsulan los datos – agrupan datos y procesos logicamente relacionados

Sistemas de Gran Porte



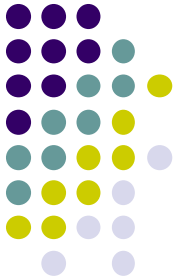
- Características
  - Gran número de entidades de computación y lineas de código.
  - Equipo de programadores
  - Código distribuído en varios archivos fuente
  - Conveniente no recompilar partes no alteradas del programa.

Sistemas de Gran Porte



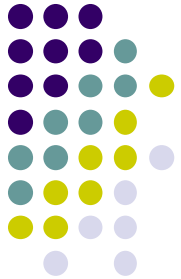
- Módulo
  - Unidad que puede ser compilada separadamente
  - Propósito único.
  - Interfaz apropiada con otros módulos.
  - Reutilizable y Modificable.
  - Puede contener uno o más tipos, variables, constantes, funciones, procedimientos.
  - Debe identificar claramente su objetivo y como lo alcanza.

Abstración



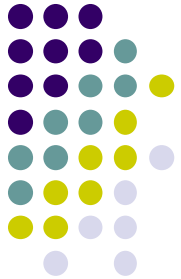
- Fundamental para la Modularización
- Selección de lo que debe ser representado
- Posibilita el trabajo en niveles de implementación y uso
- Uso diseminado en la computación

Abstracción



- Ejemplos de uso en la computación
  - Comandos del SO
  - Assemblers
  - LPs
  - Programa de Reservas
- Modos
  - LP es abstracción sobre el hardware
  - LP ofrece mecanismos para el programador crear sus abstracciones sobre el problema
  - El segundo modo fundamenta la modularización

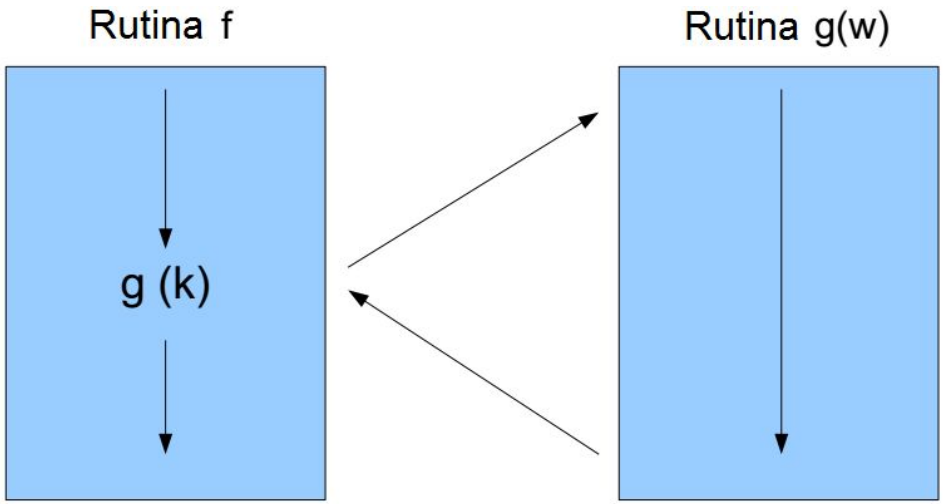
Tipos de Abstracciones



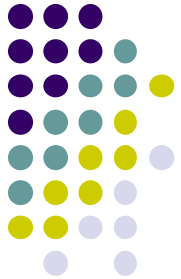
- Abstracciones de Processos
  - Abstracciones sobre el flujo de control del programa
  - Suprogramas o rutinas– funciones de la biblioteca estándar de C (*printf*)
- Abstracciones de Datos
  - Abstracciones sobre las estructuras de datos del programa
  - Tipos de Datos – tipos de la biblioteca estándar de C (*FILE*)

Abstracciones de Procesos

- Subprogramas o rutinas
  - Permiten segmentar el programa en varios bloqueos lógicamente relacionados permitiendo reuso trechos de código que operan sobre datos diferenciados
  - Modularizaciones efectuadas con base en el tamaño del código (mito antiguo) poseen baja calidad
  - Propósito único y claro facilita legibilidad, depuración, mantenibilidad y reutilización.

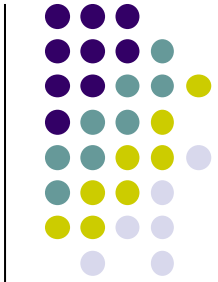


Perspectivas del Usuario y del Implementador del Subprograma



- Usuario
  - Interesa lo que el subprograma hace
  - Como usar es importante
  - Como lo hace es poco importante o no es importante
- Implementador
  - Importante es como el subprograma realiza la funcionalidad

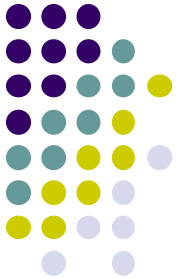
Perspectivas del Usuário e del Implementador Sobre Función



```
int factorial(int n) {
    if (n<2) {
        return 1;
    } else {
        return n * factorial (n - 1);
    }
}
```

- Usuario
  - Función factorial es mapeamiento de n para n!
- Implementador
  - Uso de algoritmo recursivo

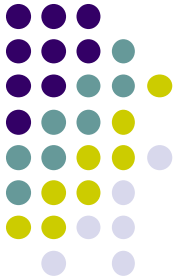
Perspectivas del Usuario y del Implementador Sobre Procedimento



```
void ordena (int numeros[50]) {
   int j, k, aux ;
   for (k = 0; k < 50; k++) {
    for (j = 0; j < 50; j++) {
       if (numeros[j] < numeros[j+1]) {
          aux = numeros[j];
          numeros[j] = numeros[j+1];
          numeros[j+1] = aux;
       }
    }
   }
}
```

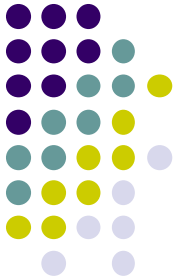
- Usuario
  - Ordenación de vector de enteros
- Implementador
  - Método de la burbuja

Parámetros



- Los parámetros son un canal de comunicación entre una rutina llamadora y una llamada
- En la rutina llamadora, el parámetro es técnicamente llamado de argumento o parámetro real.
- En la rutina llamada, el parámetro es llamado de parámetro formal
- Tipos comunes de paso de parámetros: Valor, Referencia, Valor/Resultado, Nombre.

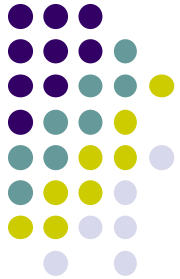
Parámetros



¿Cuál es el impacto de la ausencia de parámetros?

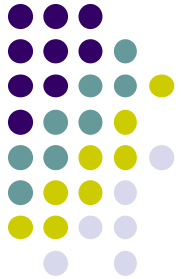
```
int altura, largo, ancho;
int volume () { return altura * largo * ancho; }
main() {
    int a1 = 1, l1 = 2, c1 = 3, a2 = 4, l2 = 5, c2 = 6;
    int v1, v2;
    altura = a1;
    largo = l1;
    ancho = c1;
    v1 = volume();
    altura = a2;
    largo = l2;
    ancho = c2;
    v2 = volume();
    printf ("v1: %d\nv2: %d\n", v1, v2);
}
```

Parámetros



- Ausencia reduce
  - Facilidad de Escritura
    - Necesario incluir operaciones para atribuir los valores deseados a las variables globales
  - Legibilidad
    - En llamada de `volume()` no existe cualquier mención a la necesidad de uso de los valores de las variables altura, largura y comprimento
  - Confiabilidad
    - No exige que sean atribuidos valores a todas las variables globales utilizadas en volume

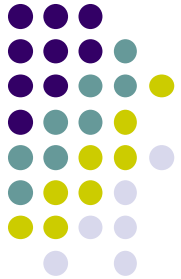
Parámetros



- Resuelven esos problemas

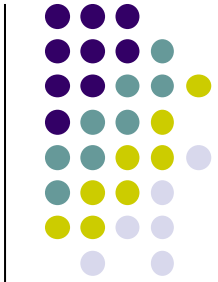
```
int volume (int altura, int largura, int comprimento) {
   return altura * largura * comprimento;
}
main() {
   int a1 = 1, l1 = 2, c1 = 3, a2 = 4, c2 = 5, l2 = 6;
   int v1, v2;
   v1 = volume(a1, l1, c1);
   v2 = volume(a2, l2, c2);
   printf ("v1: %d\nv2: %d\n", v1, v2);
}
```

Parámetros Reales, Formales y Argumentos



- Parámetro formal
  - Identificadores listados en la cabecera del subprograma y usados en su cuerpo
- Parámetro real
  - Valores, identificadores o expresiones utilizados en la llamada del subprograma
- Argumento
  - Valor pasado del parámetro real para el parámetro formal

Parámetros Reales, Formales y Argumentos



Parámetro formal

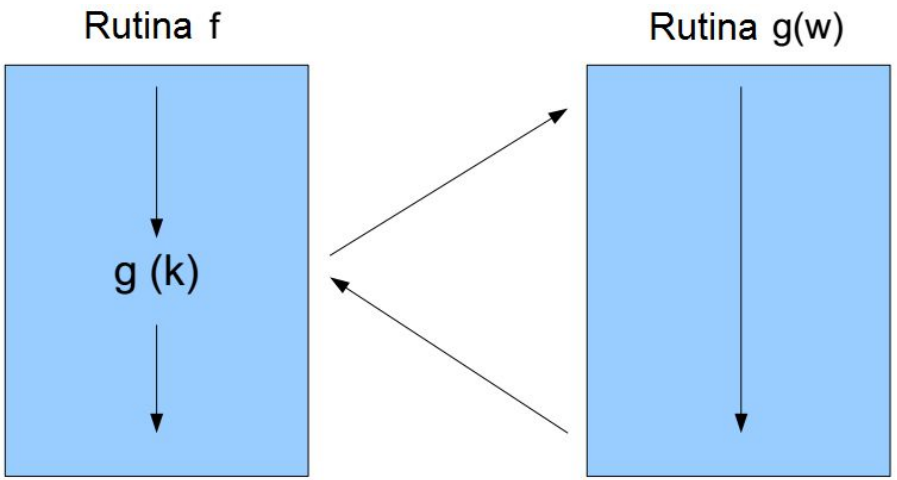
```
float area (float r) {
    return 3.1416 * r * r;
}
main() {
    float diametro, resultado;
    diametro = 2.8;
    resultado = area (diametro/2);
}
```

Parámetro real = diámetro / 2
Argumento = 1.4

- Correspondencia entre parámetros reales y formales puede ser
  - Posicional
  - Por palabras clave

Paso de Parámetros por Valor

- En este modo, en la transferencia de ejecución de la rutina llamadora para la llamada es realizada una **copia** del valor del argumento para el parámetro.
- A rutina llamada “no consigue modificar el valor” del argumento pasado
- Este modo provee apenas valores de entrada para una dada rutina llamada (unidireccional de entrada variable)
- Facilita la recuperación del estado del programa en interrupciones inesperadas
- Ejemplo: paso de parámetro en C, Java.




```c
void g (int w) {
    w++;
    printf("%d\n", w);
}

int main() {
    int k = 1;
    g(k);
    printf("%d\n", k);
    return 0;
}
```

Paso de Parámetros por Referencia

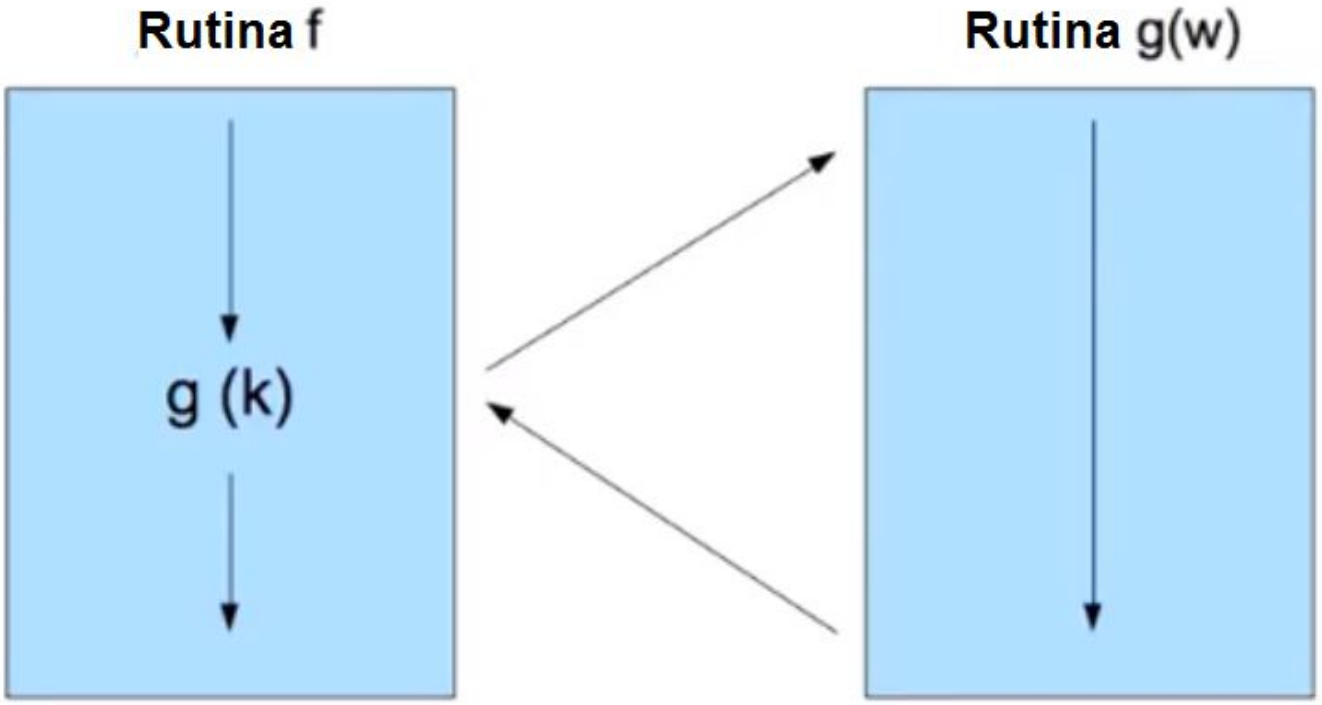
- En este modo, en la transferencia de ejecución de la rutina llamadora para la llamada el parámetro pasa a ser un “sinónimo” del argumento.
- Cualquier modificación del parámetro altera el argumento que fue informado en la función llamadora
- Más eficiente por no involucrar copia de datos
- Puede ser ineficiente en implementación distribuída

```c
#include "stdio.h"

void g (int *w) {
    (*w)++;
    printf("%d\n", *w);
}

int main() {
    int k = 1;
    g(&k);
    printf("%d\n", k);
    return 0;
}
```

```c
#include "stdio.h"

void g (int *w) {
    (*w)++;
    printf("%d\n", *w);
}

void g2 (int &w) {
    w++;
    printf("%d\n", w);
}

int main() {
    int k = 1;
    //g(&k);
    g2(k);
    printf("%d\n", k);
    return 0;
}
```

Paso de Parámetros por Resultado

- Modo dual al paso por valor
- O sea, este modo permite el retorno de valores de salida actualizados por una dada rutina llamada
- El valor inicial del argumento pasado es desconsiderado



Paso de Parámetros por Resultado

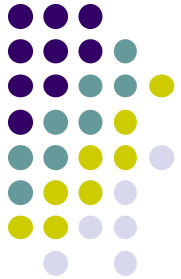
La copia no es en la ida es en la vuelta

```
2  #include "stdio.h"
3
4  void g (out int w) {
5      w = 1;
6      printf("%d\n", w);
7  }
8
9  int main() {
10     int k/* = 1*/;
11     g(k);
12     printf("%d\n", k);
13     return 0;
14 }
15
```

Paso de Parámetros por Valor- Resultado

- Combina los modos valor y resultado
- Difiere del paso por referencia, pues las áreas de memoria para argumento y parámetro son distintas entre las rutinas llamadora y llamada
- La ejecución entre este modo y el de referencia podría ser diferente, por ejemplo en un programa multithreaded

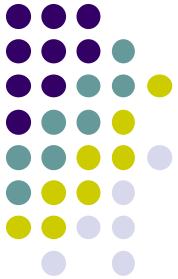
Paso de Parámetros por Nombre



- Típicamente utilizado cuando queremos pasar un argumento que será posteriormente evaluado.
- Ejemplo a seguir, en Algol, extraído del link abajo:
- http://en.wikipedia.org/wiki/Jensen%27s_Device
- Imagine que queramos computar la siguiente fórmula:

$$H_{100} = \sum_{i=1}^{100} \frac{1}{i}$$

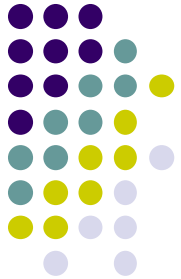
Paso de Parámetros por Nombre



En Algol, esto puede ser hecho de la siguiente forma:

```
begin
        integer k;
        real procedure sum (i, lo, hi, term);
          value lo, hi;
          integer i, lo, hi;
          real term;
          comment term pasado por nombre, así como i;
        begin
          real temp;
          temp := 0;
          for i := lo step 1 until hi do
            temp := temp + term;
          sum := temp
        end;
        comment observe la correspondencia entre la notación matemática y la llamada a la función
        sum;
        print (sum (k, 1, 100, 1/k))
end
```

Paso de Parámetros por Nombre

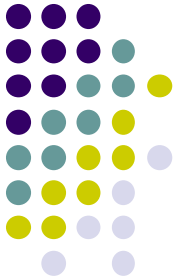


Los argumentos i y 1/i son pasados por nombre, lo que permite evaluar la expresión sólo dentro de la función, cuando se hace referencia a ella.

Para que la función de suma calcule otras fórmulas, basta pasar la ecuación correspondiente

- i y 1/(1 - i*i)
- k y k*k
- j y f(j)

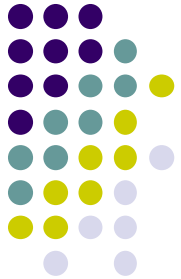
Paso de Parámetros por Nombre



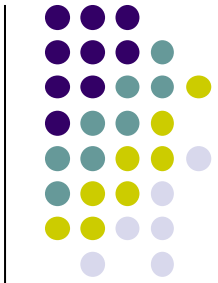
//Simulación

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "stdarg.h"
float frac(int);
float harmonico(int, int, float f(int));
float frac(int k) { return (1.0/k); }
float frac2(int k) { return (1.0/(k*k)); }
float harmonico(int lo, int hi, float f(int)) {
     float temp = 0;
     int i;
     for (i=lo; i<=hi; i++)
          temp = temp + f(i);
     return temp;
}
main() {
     printf("Valor del harmonico: %f\n", harmonico(1, 100, frac2));
}
```

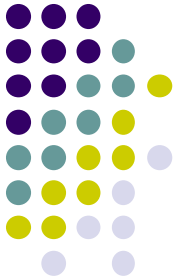
Mecanismos de Paso



- C ofrece apenas paso por copia
- C++ y PASCAL ofrecen mecanismos de copia y referencia
- ADA usa copia para primitivos y referencia para algunos tipos
  - Otros tipos son definidos por la implementación del compilador y pueden ser copia o referencia

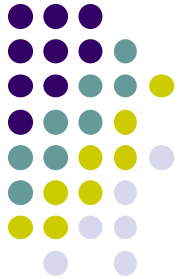


Momento de paso



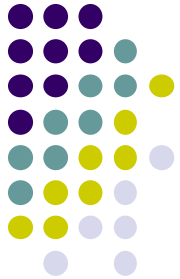
- Normal (`eager`)
  - Evaluación en la llamada del subprograma
- Por nombre (`by name`)
  - Evaluación cuando parámetro formal es usado
- Perezosa (`lazy`)
  - Evaluación cuando parámetro formal es usado por primera vez
- Mayoria de los LPs (tales como, C, PASCAL, JAVA y ADA) adopta modo normal

Momento del paso



```
int caso (int x, int w, int y, int z) {
    if (x < 0) return w;
    if (x > 0) return y;
    return z;
}
caso(p(), q(), r(), s());
```

Momento del paso



- Evaluación normal
  - Evaluación desnecesaria de funciones en caso
  - Puede reducir eficiencia y flexibilidad
- Evaluación por nombre
  - Solamente una de q, r o s seria evaluada
  - Problemas
    - p podria ser evaluada dos veces
    - Si p produciese efectos colaterales (como en un iterador)
- Evaluación Perezosa
  - Única ejecución de p y somente una de q, r o s