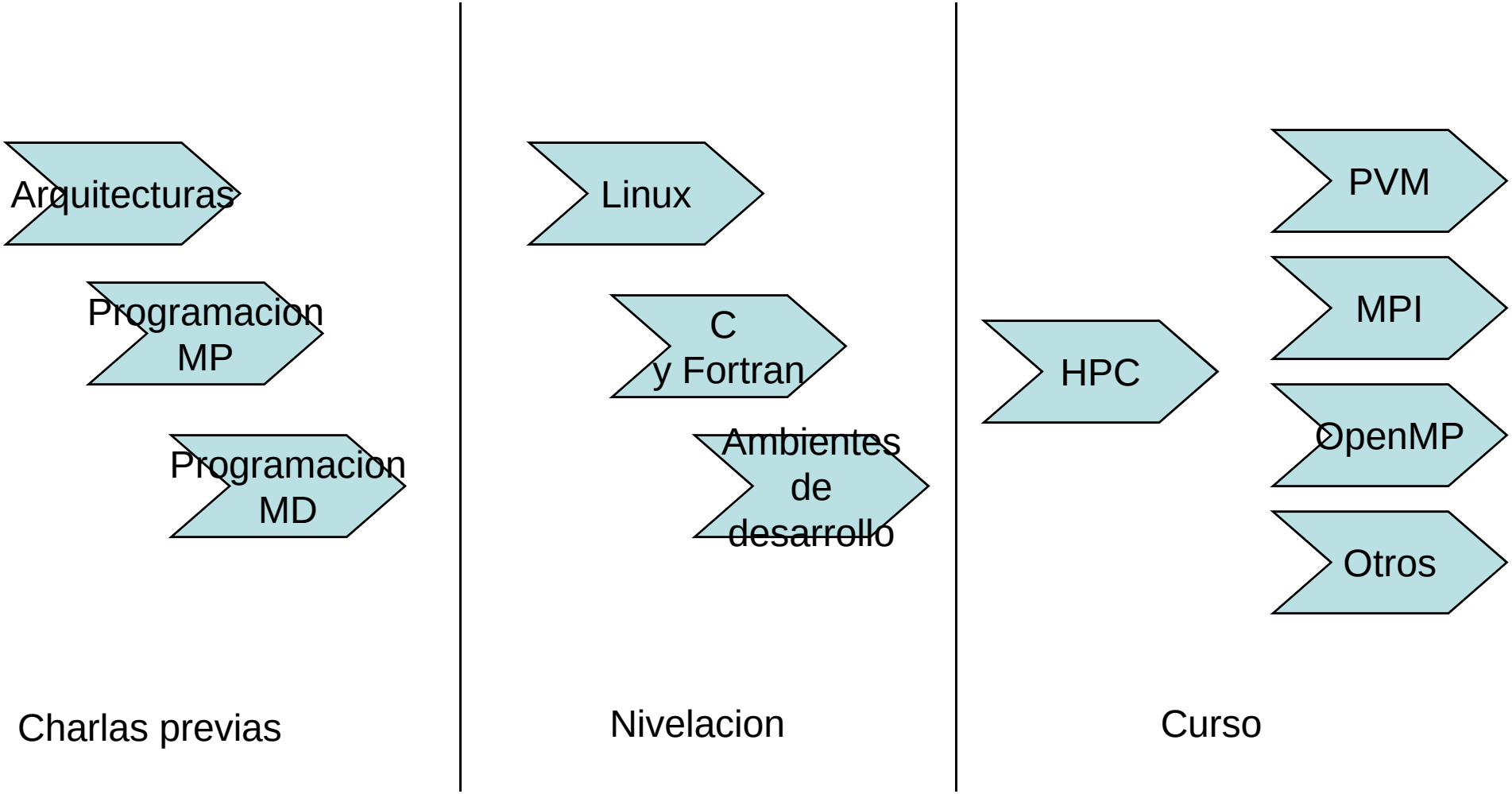
Actualizacion y nivelacion de conocimientos de computación

orientado para futuros participantes del curso de HPC

Julio 2009

Contexto


Arquitecturas
Programacion MP
Programacion MD
Linux
C y Fortran
Ambientes de desarrollo
HPC
PVM
MPI
OpenMP
Otros
Charlas previas
Nivelacion
Curso

Preguntas ?

Agenda

- **DIA 1 (8 de julio): Ambiente LINUX**
  Bibliografia: el entorno de programacion UNIX, de Kernighan y Pike
- **DIA 2 (15 de julio): Los lenguajes de programacion C y FORTRAN**
  Bibliografia: el lenguaje de programacion C, de Kernighan y Ritchie
- **DIA 3 (22 de julio): Ambiente de desarrollo C en LINUX**

Agenda

- **DIA 2 (15 de julio): Los lenguajes de programacion C y FORTRAN**
  - **Estructuras de control (if, for, loops, etc)**
  - **Tipos básicos de datos (enteros, double, etc)**
  - **Memoria dinámica (malloc y free)**
  - **Encapsulamiento de código (funciones, pasaje de parametros)**
    Estructuras de datos (arrays, estructuras, etc)
  - **Creación de procesos (fork)**
  - **Introducción al Fortran**

Lenguaje C

Hello World !!!

```
#include <stdio.h>

main() {
  printf ( “hello world\n” );
}
```

Bibliotecas existentes (C-library)

#include < math.h>

< stdio.h> -> defining I/O routines
< ctype.h> -> defining character manipulation routines
< string.h> -> defining string manipulation routines
< math.h>   -> defining mathematical routines
< stdlib.h> -> defining number conversion, storage allocation
and similar tasks
< stdarg.h> -> defining libraries to handle routines with variable
numbers of arguments
< time.h>   -> defining time-manipulation routines

< assert.h> -> defining diagnostic routines
< setjmp.h> -> defining non-local function calls
< signal.h> -> defining signal handlers
< limits.h> -> defining constants of the int type
< float.h> -> defining constants of the float type

Estructuras de control

```
While ( I <= 5 ) {
    :
}

for (i=1; i<=5; i=i+1) {
    :
}

if ( condicion1 ) {
} else if ( condicion2 ) {
} else {
}
```

Estructuras de control

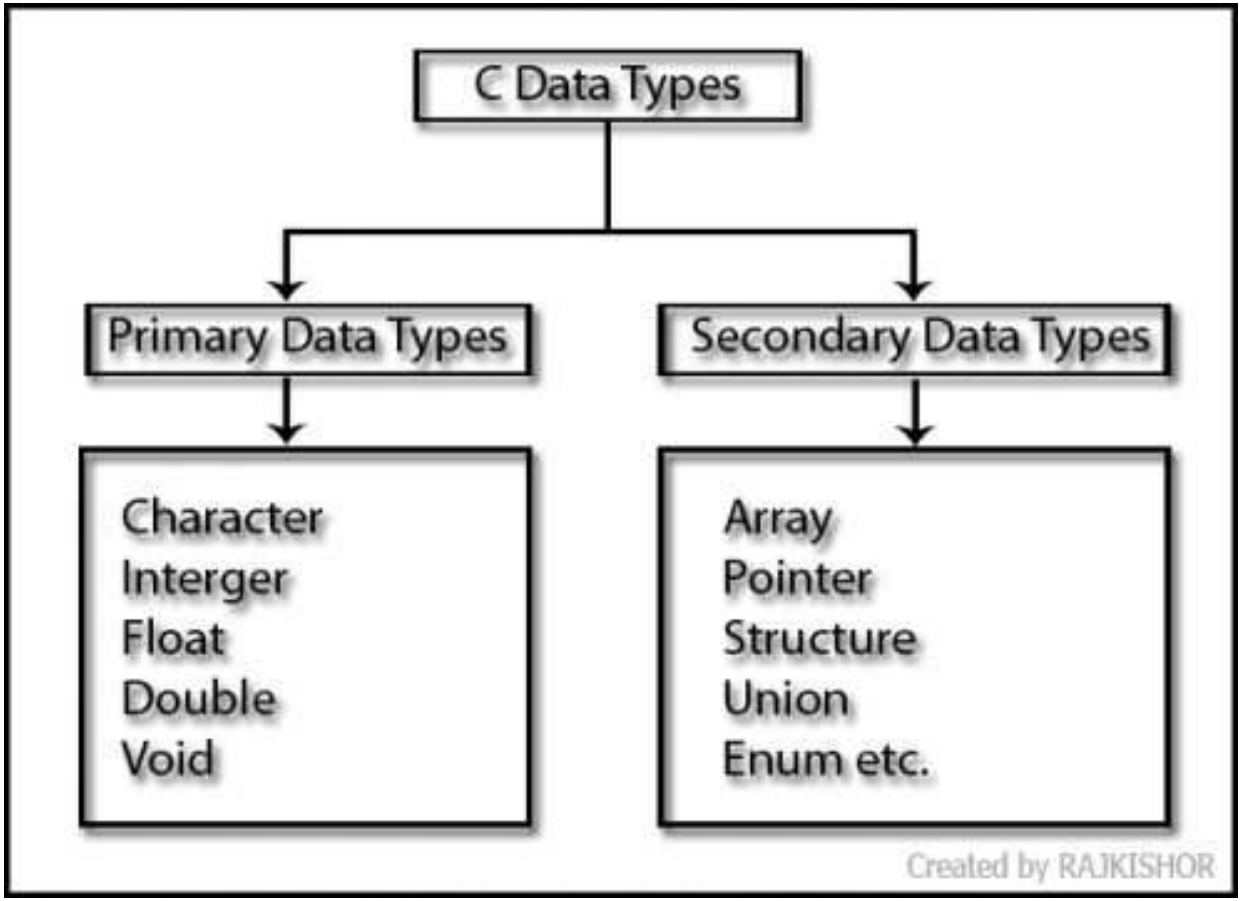
```
switch ( a ) {
    case 1 : {
      :
      break;
    }
    case 2 : {
      :
      break;
    }
    default: {

    }
}
```

Estructuras de control

Salidas forzadas:

```
  Break;

  label:
  goto label;
```

Tipos de datos



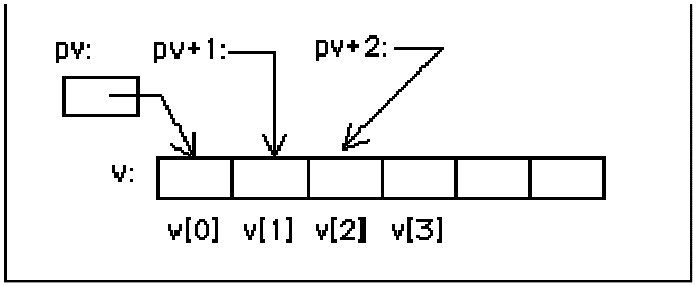
Tipos basicos de datos

Variable Type	Keyword	Bytes Required	Range
Character	char	1	-128 to 127
Unsigned character	unsigned char	1	0 to 255
Integer	int	2	-32768 to 32767
Short Integer	short int	2	-32768 to 32767
Long Integer	long int	4	-2,147,483,648 to 2,147,438,647
Unsigned Integer	unsigned int	2	0 to 65535
Unsigned Short Integer	unsigned short int	2	0 to 65535
Unsigned Long Integer	unsigned long int	4	0 to 4,294,967,295
Float	float	4	1.2E-38 to
Double	double	8	2.2E-308 to
Long Double	long double	10	3.4E-4932 to 1.1E+4932

Estructuras de datos

- **Vectores y matrices**:

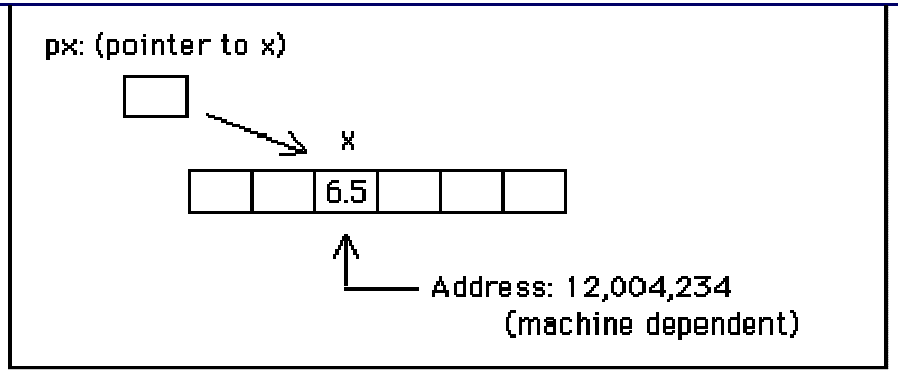
```
Tipo nombre[dimension1][dimension2]…

Char nombre[100]
Int matriz[10][10]
```



- **Estructuras**

```
struct alumno {
        char nombre[20];
        int edad;
        int curso;
  };


  Struct alumno pepe;
  pepe.nombre = “juan”;
  pepe.edad = 20;
  pepe.curso =1234;

  Struct alumno clase[200];
```

Punteros



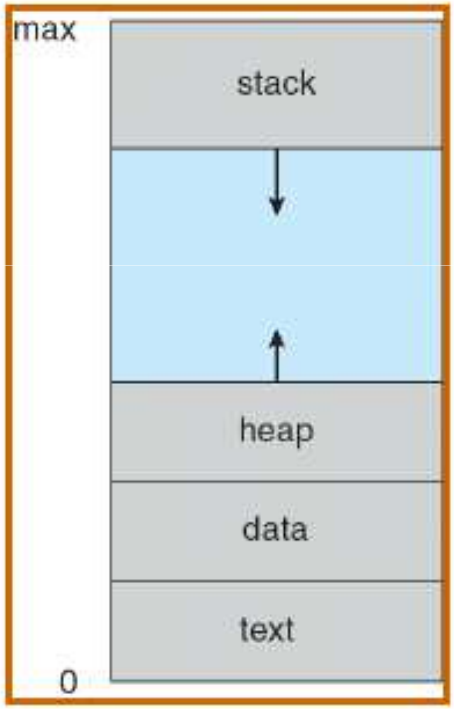

```
float x;
float* px;

x = 6.5;
px = &x;   // operador “direccion de” o referenciacion

float xx;
xx=*px;     // operador “lo apuntado por”
                o dereferenciacion
```

Uso de memoria por los procesos

Memoria de los procesos

Un proceso en memoria se constituye de varias secciones:

- Código (*text*): Instrucciones del proceso.
- Datos (*data*): Variables globales del proceso.
- ***Heap***: Memoria dinámica que genera el proceso.
- ***Stack***: Utilizado para preservar el estado en la invocación anidada de procedimientos.



Sistemas Operativos - Teórico 2009 - Procesos

Memoria dinámica

Las dos funciones que nos permiten reservar memoria son:

```
malloc (cantidad_de_memoria);
calloc (número_de_elementos, tamaño_de_cada_elemento);
```

Funcion para liberar memoria:

```
free (puntero)
```

Memoria dinámica

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

main()   {
    int     *p_int;
    float   *mat;

    p_int = (int *)  malloc(sizeof(int));
    mat   = (float *)calloc(20,sizeof(float));

    if ((p_int==NULL)||(mat==NULL))      {
                    printf ("\nNo hay memoria");
                    exit(1);
            }
    /* Aquí irían las operaciones sobre los datos */

  free(pint);
  free(mat);
}
```

Encapsulamiento de código

```
return-type function-name ( argument-list-if-necessary ) {

    Declaracion de variables locales
    Codigo de la funcion

    return return-value;

}
```

- El pasaje de parametros debe ser por referencia (direcciones de memoria) si la funcion quiere alterar los datos recibidos
- Void significa nada

Pasaje de parametros

```
#include < stdio.h>

void exchange(int a, int b);                                    /* WRONG CODE */

void main() {
   int a, b;

   a = 5;
   b = 7;
   printf("From main: a = %d, b = %d\n", a, b);

   exchange(a, b);
   printf("Back in main: ");
   printf("a = %d, b = %d\n", a, b);
}

void exchange(int a, int b) {
   int temp;

   temp = a;
   a = b;
   b = temp;
   printf(" From function exchange: ");
   printf("a = %d, b = %d\n", a, b);
}
```

Pasaje de parametros

```
#include < stdio.h>

void exchange ( int *a, int *b );

void main() {                                   /* RIGHT CODE */
   int a, b;

   a = 5;
   b = 7;
   printf("From main: a = %d, b = %d\n", a, b);

   exchange(&a, &b);
   printf("Back in main: ");
   printf("a = %d, b = %d\n", a, b);
}

void exchange ( int *a, int *b ) {
   int temp;

   temp = *a;
   *a = *b;
   *b = temp;
   printf(" From function exchange: ");
   printf("a = %d, b = %d\n", *a, *b);
}
```

Creación de procesos

Mecanismos a presentar

- Fork y wait

- **Otros mecanismos:**
  - Threads

fork()

- La primitiva fork() crea un proceso hijo del proceso invocante (“clonación” del proceso).
- Se crea una copia exacta del proceso que la invoca, pero con otro PID.
- El padre recibe el PID del hijo, y el hijo recibe 0.
- Se copia el área de datos (incluyendo archivos abiertos y sus descriptores)
- La primitiva wait() espera que los hijos terminen.

wait()

- El hijo retorna un codigo (valor) de terminacion al padre.
- Si el padre muere antes que el hijo, el hijo queda “huérfano”.
- El proceso "init" (PID=1) hereda a los huerfanos.
- Si el hijo termina pero el padre no acepta su codigo de terminación (usando wait), el proceso pasa a estado zombi (no consume recursos salvo una entrada en la tabla de procesos)
- wait retorna el PID del hijo que termino.

fork - ejemplo

```
#include <stdio.h>
/*  creación de un proceso hijo e impresión en pantalla;
    la salida se mezcla evidenciando el time slicing  */

main () {
int j, pid ;

  if ( (pid = fork ()) == 0 ) {
       for (j=0; j<1000;j++)
           printf("H");
       exit (0) ;
  }

  for (j=0; j<1000;j++)
       printf("P");
  exit (0) ;
}
```

Comunicacion entre procesos (IPC)

- **Algunas formas:**
  - **Pipes**
  - **Sockets**
  - **Shared Memory**
  - **Signals**
  - **Remote Procedure Calls**
  - **Colas de mensajes**
  - **Web Services**

Lenguaje Fortran