Tema 6: Memoria dinámica

Programación 2

Curso 2013-2014

Notas

Tema 6

Organización de la memoria
Memoria estática
Memoria dinámica
Punteros
Definición y declaración
Dirección y contenido
Declaración con inicialización
Ejercicios
Uso de punteros
Reserva y liberación de memoria
Punteros y vectores
Punteros definidos con typedef
Punteros y registros
Parámetros de funciones
Errores comunes
Ejercicios

Índice

Notas

Memoria estática

- El tamaño es fijo y se conoce al implementar el programa
- Declaración de variables

```
int i=0;
char c;
float vf[3]={1.0, 2.0, 3.0};
```

i	c	vf[0]	vf[1]	vf[2]
0		1.0	2.0	3.0
1000	1002	1004	1006	1008

Notas

Memoria dinámica

- Normalmente se utiliza para almacenar grandes volúmenes de datos, cuya cantidad exacta se desconoce al implementar el programa
- La cantidad de datos a almacenar se calcula durante la ejecución del programa y puede cambiar
- En C++ se puede hacer uso de la memoria dinámica usando punteros

Notas

Definición y declaración

- Un puntero es un número (entero largo) que se corresponde con una dirección de memoria
- En la declaración de un puntero, se debe especificar el tipo de dato que contiene la dirección de memoria
- Se declaran usando el carácter *
- Ejemplos:

```
int *punteroAEntero;
char *punteroAChar;
int *VectorPunterosAEntero[tVECTOR];
double **punteroAPunteroAReal;
```

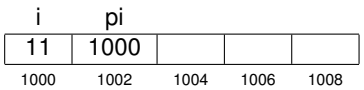
Notas

Dirección y contenido

`*x`	Contenido de la dirección apuntada por `x`
`&x`	Dirección de memoria de la variable `x`

- Ejemplo:

```
int i=3;
int *pi;
pi=&i;     // pi=direccion de memoria de i
*pi = 11; // contenido de pi=11. Por lo tanto, i = 11
```



Notas

Declaración con inicialización

- Declaración con inicialización:

```
int *pi=&i;     // pi contiene la direccion de i
```

- El puntero NULL es aquel que no apunta a ninguna variable

```
int *pi=NULL;
```

- Precaución: siempre que un puntero no tenga memoria asignada debe valer NULL.

Notas

Ejercicios

Ejercicio 1

Indica cuál sería la salida de los siguientes fragmentos de código:

```
int e1;
int *p1, *p2;
e1 = 7;
p1 = &e1;
p2 = p1;
e1++;
(*p2) += e1;
cout << *p1;
```

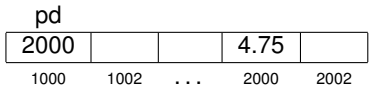
```
int a=7;
int *p=&a;
int **pp=&p;
cout << **pp;
```

Notas

Reserva y liberación de memoria (1/2)

- Los punteros pueden usarse para reservar (con new) o liberar (con delete) memoria dinámica
- Ejemplo:

```
double *pd;
pd = new double;        // reserva memoria
*pd = 4.75;
cout << *pd << endl;    // muestra 4.75
delete pd;              // libera memoria
```



- Ojo: Las variables locales y las reservadas con `new` van a zonas de memoria distintas.

Notas

Reserva y liberación de memoria (2/2)

- Cuando se usa new, el compilador reserva memoria y devuelve la dirección de inicio de ese espacio reservado
- Si no hay suficiente memoria para la reserva, new devuelve NULL
- Siempre que se reserva memoria con `new` hay que liberarla con `delete`
- Ojo: Tras hacer delete, el puntero no vale NULL por defecto
- Un puntero se puede reutilizar; tras liberar su contenido se puede reservar memoria otra vez con new

Notas

Punteros y vectores (1/2)

- Los punteros también pueden usarse para crear vectores o matrices
- Para reservar memoria hay que usar corchetes y especificar el tamaño
- Para liberar toda la memoria reservada es necesario usar corchetes
- Ejemplo:

```
int *pv;
int n=10;
pv=new int[n]; // reserva memoria para n enteros
pv[0]=585; // usamos el puntero como si fuera un vector
delete [] pv; // liberar la memoria reservada
```

Notas

Punteros y vectores (2/2)

- Los punteros se pueden usar como accesos directos a componentes de vectores o matrices
- Ejemplo:

```
int v[tVECTOR];
int *pv;
pv = &(v[7]);
*pv = 117; // v[7] = 117;
```

Notas

Punteros definidos con typedef

- Se pueden definir nuevos tipos de datos con typedef:

```
typedef int entero; // entero es un tipo, como int
entero a,b; // int a,b;
```

- Para facilitar la claridad en el código, suelen definirse los punteros con typedef:

```
typedef int *punteroAEntero;
typedef struct {
    char c;
    int i;
} Registro, *PunteroRegistro;
```

Notas

Punteros y registros

- Cuando un puntero referencia a un registro, se puede usar el operador -> para acceder a sus campos

```
typedef struct {
   char c;
   int i;
} Registro, *PunteroRegistro;

PunteroRegistro pr;
pr = new Registro;
pr->c = 'a';  // (*pr).c = 'a';
```

Notas

Parámetros de funciones (1/2)

- Paso de parámetros a funciones

```
void f (int *p) {  // Por valor
  *p=2;
}

void f2 (int *&p) { // Por referencia
  int num=10;
  p=&num;
}

int main() {
  int i=0;
  int *p=&i;
  f(p);       // Llamada a funciones
  f2(p);
}
```

Notas

Parámetros de funciones (2/2)

- Paso de parámetros a funciones usando typedef

```
typedef int* PInt;

void f (PInt p){  // Por valor
  *p=2;
}

void f2 (PInt &p) { // Por referencia
  int num=10;
  p=&num;
}

int main() {
  int i=0;
  PInt p=&i;
  f(p);       // Llamada a funciones
  f2(p);
}
```

Notas

Errores comunes

- Utilizar un puntero sin haberle asignado memoria o apuntando a nada

```
int *pEntero;
*pEntero = 7; // Error !!!
```

- Usar un puntero tras haberlo liberado

```
punteroREGISTRO p,q;
p = new REGISTRO;
...
q = p;
delete p;
q->num =7; // Error !!!
```

- Liberar memoria no reservada

```
int *p=&i;
delete p;
```

Notas

Ejercicios

Ejercicio 2

- Dado el siguiente registro:

```
typedef struct {
   char nombre[32];
   int edad;
} Cliente;
```

Realizar un programa que lea un cliente (sólo uno) de un fichero binario, lo almacene en memoria dinámica usando un puntero, imprima su contenido y finalmente libere la memoria reservada.

Notas