Informática I para Bachillerato

C/C++

Apuntadores

José Luis Alonzo Velázquez

CIMAT

Sesión 13

Variables en memoria

Antes que nada hay que ver guardamos las variables C/C++ es decir como utilizamos los diferentes tipos de memoria:

- Variables locales en M. PILA

Variables en memoria

Antes que nada hay que ver guardamos las variables C/C++ es decir como utilizamos los diferentes tipos de memoria:

- Variables locales en M. PILA
- Variables globales en M. ESTÁTICA

Variables en memoria

Antes que nada hay que ver guardamos las variables C/C++ es decir como utilizamos los diferentes tipos de memoria:

- Variables locales en M. PILA
- Variables globales en M. ESTÁTICA
- Variables estáticas en M. ESTÁTICA

Variables en memoria

Antes que nada hay que ver guardamos las variables C/C++ es decir como utilizamos los diferentes tipos de memoria:

- Variables locales en M. PILA
- Variables globales en M. ESTÁTICA
- Variables estáticas en M. ESTÁTICA
- Datos alocados dinámicamente en M. MONTÍCULO

M. PILA/STACK

La pila de llamadas (stack) es una estructura dinámica de datos LIFO, (una pila), es el lugar que almacena la información sobre las subrutinas activas de un programa. Esta clase de pila también es conocido como una pila de ejecución, pila de control, pila de función, o pila de tiempo de ejecución, y a menudo se describe en forma corta como “la pila”.

Básicamente

Es un área en la que las variables aparecen y desaparecen en un momento puntual de la ejecución de un programa. El funcionamiento de la pila es como un espacio de anotación temporal. Se utiliza principalmente para almacenar variables locales a las funciones. Estas variables tienen un ámbito reducido, sólo están disponibles mientras se está ejecutando la función en la que han sido definidas. En la pila se encuentran todas estas variables, y por tanto, en esa zona se está continuamente insertando y borrando variables nuevas.

Variables temporales

```
1 #include <stdio.h>
2 int main(){
3     for(int i;i<10;i++){
4         printf("Hola Mundo!!!\n");
5     }
6     return 0;
7 }
```

M. ESTÁTICA

En esta zona de memoria se almacenan todos aquellos datos que están presentes desde el comienzo del programa hasta que termina. Un ejemplo claro de esto son las variables globales.

Variables Globales

¿Qué son las variables globales y cuando las utilizamos?

Variables Globales

```
1  #include <stdio.h>
2
3  int x;
4
5  int modifica(){
6      x=3;
7      return 0;
8  }
9
10 int main(){
11     x=2;
12     printf("El valor de x es: %d\n",x);
13     modifica();
14     printf("El valor de x es: %d\n",x);
15     return 0;
16 }
```

Nota sobre Variables Globales:

Se pueden acceder desde otros archivos compilados en la misma aplicación, con la palabra llave **extern**: En file1.cpp: int myGlobalVariable; En file2.cpp :

```
1 extern int myGlobalVariable;
```

Variables Globales

Cuando hay conflicto entre nombres de variables locales y de variables globales, a priori es la variable local que es considerada. Para utilizar explícitamente la variable global, se puede usar la notación ::var (también es valido para funciones/métodos, dentro de clases):

Ejemplo

```
1 int x=5;
2 int func(int n){
3     int x ;
4     x = 1 ;
5     ::x = n+1;
6 }
```

M. MONTICULO/HEAP

Esta zona contiene memoria disponible para que se reserve y libere en cualquier momento durante la ejecución de un programa. No está dedicada a variables locales de las funciones como la pila, sino que es memoria denominada "dinámica" para estructuras de datos que no se saben si se necesitan, e incluso tampoco se sabe su tamaño hasta que el programa está ejecutando.

Alocación dinámica

La alocación dinámica permite reservar pedazos de memoria en una zona especifica de la memoria (el montículo, o heap), al momento de la ejecución, según las necesidades.
Espacio reservado a través de un apuntador inicializado por funciones como malloc.
Espacio no liberado automáticamente (como lo de la pila) : uso de funciones explícitas de liberación, como free.

Apuntadores(punteros)

Un puntero es una variable que contiene una dirección de memoria. Normalmente, esa dirección es la posición de otra variable de memoria. Si una variable contiene la dirección de otra variable, entonces se dice que la primera variable apunta a la segunda.

Apuntadores(punteros)

Un puntero es una variable que contiene una dirección de memoria. Normalmente, esa dirección es la posición de otra variable de memoria. Si una variable contiene la dirección de otra variable, entonces se dice que la primera variable apunta a la segunda.

Sintaxis

TIPO * nombre_puntero ;

Apuntadores(punteros)

Un puntero es una variable que contiene una dirección de memoria. Normalmente, esa dirección es la posición de otra variable de memoria. Si una variable contiene la dirección de otra variable, entonces se dice que la primera variable apunta a la segunda.

Sintaxis

TIPO * nombre_puntero ;

Ejemplo

```
char *pchar;
```

Operadores de memoria

Existen dos operadores especiales de punteros: & y *. El operador de dirección (&) devuelve la dirección de memoria de su operando. El operador de indirección (*) devuelve el contenido de la dirección apuntada por el operando.

Asignación de punteros

Como en el caso de cualquier otra variable, un puntero puede utilizarse a la derecha de una declaración de asignación para asignar su valor a otro puntero. Por ejemplo:int x;

```
1 int *p1,*p2;
2 p1=&x;
3 p2=p1;
```

Observación

Tanto p1 como p2 apuntan a x.

Alocación dinámica en C

Típicamente, para un arreglo de 100 enteros:

```
1 int isize = 100;
2 int *i_ptr = (int *)malloc(isize *sizeof(int));
3 . . .
4 free(i_ptr) ;
```

free necesario para evitar fugas de memoria.

Alocación dinámica en C++

Otro sistema para la alocación dinámica, tal vez mas intuitivo que usar, con las palabras llaves new y delete :

```
1 int isize = 100;
2 int * i_ptr = new int [isize];
3 . . .
4 delete[ ] i_ptr;
```

delete necesario para evitar fugas de memoria.

Asignación ()

Tenemos que tener cuidado con lo siguiente

```
1 {
2 int* ip = 0;
3 ip = new int;
4 int* jp = new int(13);
5 ...
6 delete ip;
7 delete jp;
8 }
```

Alocación dinámica en C

Típicamente, para un arreglo de 100 enteros:

```
1 int isize = 100;
2 int *i_ptr = (int *)malloc(isize *sizeof(int));
3 . . .
4 free(i_ptr) ;
```

free necesario para evitar fugas de memoria.

Aquí es justo donde aplica la frase “donde hay un gran poder aplica hay una gran responsabilidad!!!”.

Alocación dinámica en C++

Otro sistema para la alocación dinámica, tal vez mas intuitivo que usar, con las palabras llaves new y delete :

```
1 int isize = 100;
2 int * i_ptr = new int [isize];
3 . . .
4 delete[ ] i_ptr;
```

delete necesario para evitar fugas de memoria.

Asignación ()

Tenemos que tener cuidado con lo siguiente

```
1 {
2 int* ip = 0;
3 ip = new int;
4 int* jp = new int(13);
5 [...]
6 delete ip;
7 delete jp;
8 }
```

¿Qué es una matriz?

Se llama matriz de orden "m x n" a un conjunto rectangular de elementos a_{ij} dispuestos en m filas y en n columnas. El orden de una matriz también se denomina dimensión o tamaño, siendo m y n números naturales.

Ejemplo de una suma de matrices

```
1  void funcion_suma(int A[2][2],int B[2][2]){
2      int C[2][2];
3      for(int i=0;i<2;i++){
4          for(int j=0;j<2;j++)
5              C[i][j]=A[i][j]+B[i][j];
6      }
7      for(i=0;i<2;i++){
8          for(j=0;j<2;j++)
9              A[i][j]=C[i][j];
10     }
11     return;
12 }
```

Ejemplo de una suma de matrices

```
1 void suma matrices(float A[2][2], float B[2][2], float
      R[2][2]){
2     //rutina que suma las matrices A y B y guarda el
      resultado en R
3     // R = A + B
4     int i,j;
5     for(i=0;i<2;i++)
6         for(j=0;j<2;j++)
7             R[i][j]=A[i][j]+B[i][j];
8 }
```

Problema

Crear un arreglo dinámico de 100 elementos que guarde 100 parejas de puntos (x,y).

 Como Programar en C/C++, Deitel (Prentice Hall), 2da Edición.

 Programming Principles and Practice Using C++, Bjarne Stroustrup.

 http://www.codeblocks.org

 http://www.wxwidgets.org