Programación y Análisis de Algoritmos

Dr. Norberto Alejandro Hernández Leandro

CIMAT-Unidad Monterrey

Septiembre 2020

Contenido

- Estructuras de control
  - Condicionales
  - Ciclos
- Funciones
- Estructuras de datos
- Arreglos estáticos

Estructuras de control

If

```
1          if(condición 1){
2              Bloque de instruciones 1
3          }else if (condición 2){
4              Bloque de instrucciones 2
5          }
6          ...
7          else if (condición n){
8              Bloque de instrucciones n
9          }else{
10             Bloque final de instrucciones
11         }
```

Si los bloques de instrucciones sólo tienen una instrucción, entonces se pueden obviar las llaves.

Estructuras de control

Switch

```
1   switch (expresión){
2       case valor1 :
3           Bloque de instrucciones 1
4           break;
5       case valor2 :
6           Bloque de instrucciones 2
7           break;
8       ...
9       case valorn :
10          Bloque de instrucciones n
11          break;
12      default :
13          Bloque de instrucciones por defecto
14          break;
15  }
```

Estructuras de control

For

- Recibe una instrucción de entrada, una condición de paro, y una instrucción de incremento.
- Las tres operaciones son opcionales.
- Si la instrucción de paro no se establece, el ciclo no termina.

```
1         for(int i=0; i<5; i++)
2             cout << i << endl;
```

Estructuras de control

While

Recibe una condición de paro, que mientras se siga cumpliendo el algoritmo se mantendrá dentro del ciclo.

```
1                   i=2;
2                   while(i<5){
3                       cout << i++ << endl;
4                   }
```

- No se entra al ciclo mientras la condición no se cumpla.

```
1                   i=2;
2                   do{
3                       cout << i++ << endl;
4                   }while(i<5);
```

- Entra en el ciclo en la primera iteración y no sigue si la condición no es respetada.

Estructuras de control

Saltos Break/Continue

- La instrucción break permite interrumpir el ciclo.

```
1                   i=2;
2                   do{
3                       cout << i++ << endl;
4                       if(i==6)
5                           break;
6                   }while(i<100);
```

- La instrucción continue permite interrumpir una iteración del ciclo.

```
1                   for(int i=0; i<5; i++){
2                       if(i%2==1)
3                           continue;
4                       cout << i << endl;
5                   }
```

Variables y constantes

- Las variables son objetos que pueden o no cambiar su valor durante el código.

```
1                int a=1;
2                a=2;            //Sin problemas
```

- Las constantes se definen mediante la palabra reservada const, y define un objeto que sólo sirve de lectura.

```
1                const int b=1;
2                b=2;            //Error: la variable sólo es
3                                //de lectura y no se le puede
4                                //cambiar el valor.
```

- Las constantes también pueden definirse de manera global mediante la instrucción #define

Variables estáticas

Las variables estáticas se definen mediante la palabra reservada static, y definen variables que mantienen su valor durante toda la ejecución del programa.

```
1          void foo(){
2              static int count = 0;
3              cout << count++ << endl;
4          }
5          int main(){
6              for(int i=0; i<5; i++){
7                  foo();
8              }
9              return 0;
10         }
```

Alcance de las variables

Las variables definidas en un código tienen un alcance de uso en el cual están definidas.

```
1          void foo(){
2              int a = 0;
3              if(a<3){
4                  int b = 0;
5                  b = 2;
6              }
7              a++;
8              b++;
9          }
```

Funciones

- Las funciones son porciones de código que realizan una determinada operación.
- Las funciones recursivas son aquellas que, dentro de su código, se mandan a llamar a sí mismas.

```
1    void foo(Argumentos){
2        Bloque de código
3        return;
4    }
```

- Al igual que los saltos en los ciclos, el return se puede utilizar para interrumpir la ejecución de la función.

Funciones

```
1   int foo(int a, int b){
2       return a+b;
3   }
4
5   int bar(int c, int d){
6       return c-d;
7   }
8
9   int main(){
10      int (*func)(int, int) = &foo;
11      int result = func(2,2);
12      cout << result << endl;
13
14      func = &bar;
15      result = func(2,2);
16      cout << result << endl;
17
18      return 0;
19  }
```

Estructuras

- Es un tipo de dato que agrupa un conjunto de datos simples en un sólo objeto.
- Se define mediante la palabra reservada struct

```
1          struct nombre{
2              tipo1 dato_1;
3              tipo2 dato_2;
4              ...
5              tipon dato_n;
6          };
```

Estructuras

- Para acceder a algún elemento de la estructura se utiliza un punto
- Si se define un apuntador de la estructura creada, los elementos pueden se accesados mediante "->"

```
1                nombre Data1;
2                cout << Data1.dato_1 << endl;
3                nombre *Data2;
4                cout << Data2->dato_n << endl;
```

¡Ojo!

En algunas versiones del compilador de C/C++

```
1        nombre Data;              //Incorrecto
2        struct nombre Data;       //Correcto
```

Solución:

```
1        typedef struct nombre{
2            tipo1 dato_1;
3            tipo2 dato_2;
4            ...
5            tipon dato_n;
6        };
```

Sobrecarga de operadores

```
1   struct par{
2       int x;
3       int y;
4   };
5
6   int main(){
7       par p1 = {1,2}, p2 = {3,4}, p3;
8       p3 = p1+p2;                 //Error, el compilador no
9                                   //sabe operar estructuras
10  }
```

Solución:

```
1   par operator + (const par p1, const par p2){
2       par p3={p1.x+p2.x, p1.y+p2.y};
3       return p3;
4   }
```