BLOCKS

Blocks are a way of solving the scoping problem. A block is a program region containing definitions of variables and that delimits the regions where these definitions apply. In C programming language, a block is created using a pair of curly braces. The beginning of the block is denoted by an open curly brace '{' and the end is denoted by a closing curly brace '}'. The block collects statements together into a single compound statements. The C code below shows two blocks. One of them defines the scope of the main function while the other (an inner block) creates a new scope inside this function. It is possible to see the definition of two variables with the same name, but inside distinct blocks.

```
#include <stdio.h>
int main() {
  int n = 1;
  {
    int n = 2;
    printf("%d\n", n);
  }
  printf("%d\n", n);
}
```

The scope of a definition is the block containing that definition, from the point of definition to the end of the block, minus the scopes of any redefinitions of the same name in interior blocks. So, the scope of the variable 'n', defined in the line 3, begins at that line and ends at line 9.

But, because there is a new definition of a variable named 'n' inside this block, the variable 'n' defined outside the inner block becomes hidden from line 5 to line 7. The same source code is presented below with the distinction of the visibility of the two variables named 'n'. The scope visibility of the first definition is represented by the letter 'A' while the scope visibility of the second definition is represented by the letter 'B'.

```
#include <stdio.h>
int main() {
  int n = 1;              A
  {                       A
    int n = 2;            B
    printf("%d\n", n);    B
  }                       B
  printf("%d\n", n);      A
}                         A
```

Many different constructs serve as blocks in different languages. The term block-structured is commonly referred to languages that use some kind of block to delimit scope. Nowadays, the majority of programming languages are block-structured and few people bother to make this distinction.

The ML programming language uses the let construct to define a block. The following source code shows an example of it use. The let construct contains definitions (located before the keyword 'in') and a region where those definitions apply (from the point of each definition to the final end). The letters 'A' and 'B' distinct the two scopes in the source code.

```
let                       A
  val n = 1               A
  val x = n + 1           A
in                        A
  let                     B
    val n = 2             B
  in                      B
    n + x                 B
  end                     B
end                       A
```

• Blocks are user-specified local scopes for variables

• An example in C

```
{int temp;
temp = list [upper];
list [upper] = list [lower];
list [lower] = temp
}
```

• The lifetime of temp in the above example begins when control enters the block

• An advantage of using a local variable like temp is that it cannot interfere with any other variable with the same name

Implementing Blocks

Two Methods:

1. Treat blocks as parameter-less subprograms that are always called from the same location

– Every block has an activation record; an instance is created every time the block is executed

2. Since the maximum storage required for a block can be statically determined, this amount of space can be allocated after the local variables in the activation record

Implementing Dynamic Scoping

• Deep Access: non-local references are found by searching the activation record instances on the dynamic chain

- Length of the chain cannot be statically determined

- Every activation record instance must have variable names

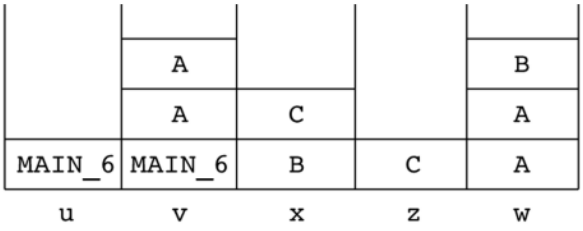
• Shallow Access: put locals in a central place

– One stack for each variable name

– Central table with an entry for each variable name

Using Shallow Access to Implement Dynamic Scoping

```
void sub3() {
 int x, z;
 x = u + v;
 …
}
void sub2() {
 int w, x;
 …
}
void sub1() {
 int v, w;
 …
}
void main() {
 int v, u;
 …
}
```



(The names in the stack cells indicate the program units of the variable declaration.)