CS 241 - WLP4 Programming Language Tutorial

WLP4 in a nutshell for C or C++ Programmers

A WLP4 program is a sequence of C++ functions (we use the terms 'procedure' and 'function' interchangeably in this document), the last of which is is the main function and is called `wain`. Functions may have any number of parameters, but the main function `wain` has two parameters. The type of the first parameter of `wain` is either `int` or `int*`. The type of the second parameter of `wain` is `int`. The type of the result of every procedure is `int`.

Declarations, control structures and statements that may be used in WLP4 are restricted to:

- `int` or `int*` *(declaration of a single int or int* variable with an unsigned integer constant or NULL initializer; all declarations in WLP4 must precede all statements and control structures; every declaration must include an integer constant or NULL initializer)*
- `if` *(must have an else clause)*
- `while`
- `return` *(must be the last statement in function)*
- `println`
- `putchar`
- `=` *(i.e. assignment)*
- `delete []` *(i.e. deallocation)*

The clauses of `if` and `while` containing statements must be enclosed in braces (i.e. {}).

Expressions may contain only variable names, integers (written in decimal without a sign), function calls, memory allocation (i.e., operator `new`), unary `&` and `*`, and the binary (two operand) versions of the following operators:

```
+ - * / % == != <= >= < >
```

Arrays of consecutive integers may be dynamically allocated using `new` and `delete []`, but their elements can be accessed only using pointer dereferences, because WLP4 does not include the C++ operator `[]`. The // notation (and only the // notation) may be used for comments.

Example WLP4 Program

```
//
// WLP4 Program to compute:
//   a^b if 0 <= a,b < 10
//   -1 otherwise
//

int wain(int a, int b) {
   int counter = 0;
   int product = 0;
   product = 0-1;  // only binary minus
   if (a >= 0) {
      if (b >= 0) {
         if (a < 10) {
            if (b < 10) {
               product = 1;
               counter = 0;
               while (counter < b) {
                  product = product * a;
                  counter = counter + 1;
               }
```

```
            } else {} // must have else
          } else {}
        } else {}
    } else {}
    return product;
}
```

Notable differences between WLP4 and C/C++

Functions

Functions in WLP4 can only have one return statement, which must be at the very end.

```
int max(int a, int b){
  if (a > b){
    return a; // cannot return here
  } else {
    return b; // or here
  }
  // must return here
}
```

Fixed for WLP4, the above code looks like this:

```
int max(int a, int b){
  int answer = 0;
  if (a > b){
    answer = a;
  } else {
    answer = b;
  }
  return answer;
}
```

Local Variable Declarations

Within a function body, all the local variables must be declared at the beginning of the function, before any other code.
Variables can only be initialized to a single number or NULL.

```
int average(int a, int b){
  int sum = 0;
  sum = a + b;
  int average = sum / 2;
  return average;
}
```

Fixed for WLP4:

```
int average(int a, int b){
  int sum = 0;
  int average = 0;
  sum = a + b;
  average = sum / 2;
  return average;
}
```

if-statements

if-statements must have an else statement (though it can be empty). The conditions for an if or a while must be singular - logical AND (&&) and OR (||) do not exist in WLP4. There is no else if in WLP4.

```
int max3(int a, int b, int c){
  int answer = 0;
  if (a > b && a > c){ // cannot use "&&"
    answer = a;
  } else if (b > a && b > c){ // or "else if"
    answer = b;
  } else {
    answer = c;
  }
  return answer;
}
```

Fixed for WLP4:

```
int max3(int a, int b, int c){
  int answer = 0;
  if (a > b) {
    if (a > c) {
      answer = a;
    } else {
      answer = c;
    }
  } else {
    if (b > c){
      answer = b;
    } else {
      answer = c;
    }
  }
  return answer;
}
```

Arrays

There are no "arrays" in WLP4, though pointers can be dereferenced to achieve the same effect.

```
int getlast(int* begin, int size){
  return begin[size - 1];
}
```

Fixed for WLP4:

```
int getlast(int* begin, int size){
  return *(begin + size - 1);
}
```

Blocks of memory can be allocated using `new` and deallocated using `delete` just like in C++. The expression `new int[n]` will allocate a block of n 32-bit words, each of which holds a single int, and return a pointer to the start of the block. You can then treat this block as an "array of ints" by accessing it through the pointer.

```
int wain(int* begin, int size){
  int* copy = NULL;
  int i = 0;
  copy = new int[size];
  while(i < size) {
    *(copy+i) = *(begin+i);
    i = i + 1;
  }
  delete [] copy;
```

```
  return 0;
}
```

The code above does nothing useful (it creates a copy of an array and then immediately deletes it) but demonstrates how to use `new` and `delete` in WLP4.

I/O

WLP4 provides standard input and output functions. For character-at-a-time I/O, it provides `putchar` and `getchar`. `putchar` is implemented as a statement, but it has the same form as it would have in C or C++: `putchar(c);`. The argument is an `int`, but only the low 8 bits are used; one byte is output to standard output. `getchar` returns an `int` of which only the low bits are set as a single byte from standard input, unless end of file was reached, in which case it returns -1.

In addition, it provides a `println` statement that prints an integer in base 10 to standard output, with a newline (i.e., on a line of its own). Like `putchar`, it's implemented as a statement, but has the same form as it would have in C or C++ as a function call: `println(i);`.