CS 241 — Spring 2020 — Assignment 9

Assignments for CS 241

← Assignment 8	Assignment 9	Assignment 10 ↓
~~Friday, July 17th at 5:00pm~~ Sunday, July 19th at 5:00pm	**Wednesday, July 29th, at 5:00 pm**	Wednesday, August 5th, at 11:59 pm

P1 • P2 • P3 • P4 • P5 • P6 • P7 • P8

In Assignments 9 and 10 you will complete the WLP4 compiler for which you wrote a scanner in Assignment 6, a parser in Assignment 7, and a context-sensitive analyzer in Assignment 8. For each problem, you will write a MIPS code generator for an increasingly larger subset of WLP4, culminating with a full WLP4-to-MIPS compiler. Each code generator will be a Racket or C++ program that takes as input a .wlp4i file and, if it conforms to the context-sensitive syntax of WLP4, produces as output a MIPS .asm file that may be assembled with cs241.binasm (or cs241.linkasm for Assignment 9 Problem 4 onwards) and then executed with mips.twoints or mips.array.

For Assignments 9 and 10, you should start with your context-sensitive analyzer from Assignment 8. However, all of the test inputs for Assignments 9 and 10 will be valid WLP4 programs. Therefore, if you did not complete Assignment 8 to reject all invalid WLP4 programs, you can still use your partial solution as a starting point for Assignments 9 and 10. In particular, if your program passes all non-blind tests for Assignment 8 Problems 1 to 5, you should have no issues doing Assignments 9 and 10. Even if you do not pass all non-blind tests, you may still be fine provided the tests you fail only involve invalid programs.

Additionally, for Assignment 9 and Assignment 10 Problems 1 and 2, the only part of Assignment 8 you need is the symbol table. However, to complete Assignment 10 you will need a working type checker. Note also that if your context-sensitive analyzer outputs a string containing ERROR to standard error for any of the test programs in Assignment 9 and 10, you will automatically fail the corresponding Marmoset test!

Testing

You must test your code generator yourself in Linux. To test your code generator, you will need .wlp4i files as generated by the parser specified in A7P5. In case you do not wish to use your own parser, you can use one that we have provided. Invoke it using the command `wlp4parse`.

Starting from a .wlp4 source file, the complete sequence of commands to generate MIPS machine language and run it is:

```
wlp4scan < foo.wlp4 > foo.scanned
wlp4parse < foo.scanned > foo.wlp4i
racket wlp4gen.rkt < foo.wlp4i > foo.asm
cs241.binasm < foo.asm > foo.mips
mips.twoints foo.mips OR mips.array foo.mips
```

OR

```
wlp4scan < foo.wlp4 > foo.scanned
wlp4parse < foo.scanned > foo.wlp4i
./wlp4gen < foo.wlp4i > foo.asm
cs241.binasm < foo.asm > foo.mips
mips.twoints foo.mips OR mips.array foo.mips
```

This can be abbreviated to

```
cat foo.wlp4 | wlp4scan | wlp4parse | racket wlp4gen.rkt | cs241.binasm > foo.mips
mips.twoints foo.mips
```

OR

```
cat foo.wlp4 | wlp4scan | wlp4parse | ./wlp4gen | cs241.binasm > foo.mips
mips.twoints foo.mips
```

The above can be made much easier to do with a shell script.

Marmoset Notes

For each question, Marmoset has public and release tests, and often has secret tests as well. The weights assigned to each question are given on the assignment.

When you release test a submission, Marmoset will only report on public and release tests. No information about secret tests will be visible until after the assignment deadline.

This means the number of marks you see on Marmoset (prior to the assignment deadline) will not necessarily match the number of marks shown on the assignment page.

Problem 1 — 4 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Write a code generator for correct WLP4 programs that conform to the syntax rules:

```
start → BOF procedures EOF
procedures → main
main → INT WAIN LPAREN dcl COMMA dcl RPAREN LBRACE dcls statements RETURN expr SEMI RBRACE
type → INT
dcls →
dcl → type ID
statements →
expr → term
term → factor
factor → ID
```

You may assume that the test input is a valid WLP4 program that satisfies the context-sensitive syntax of WLP4, and whose derivation contains only the above production rules.

Note: All that a WLP4 program conforming to this syntax can do is return the value of one of its parameters. Your MIPS output should assume that the parameter values are in $1 and $2 respectively, and return the result in $3.

Click here to go back to the top of Assignment 9.

Problem 2 — 7 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your solution for Problem 1 to handle inputs whose context-free syntax conforms to the rules for Problem 1, and in addition:

```
factor → LPAREN expr RPAREN
```

Click here to go back to the top of Assignment 9.

Problem 3 — 10 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules for Problems 1 and 2, and in addition:

```
expr → expr PLUS term
expr → expr MINUS term
term → term STAR factor
```

```
term → term SLASH factor
term → term PCT factor
factor → NUM
```

That is, you are to implement expressions with integer constants and the operators {+, -, *, /, %}

You do not need to worry about integer overflow for this problem. In particular, for multiplication, you should use `mflo` to get the result and ignore the value in the `hi` register.

Click here to go back to the top of Assignment 9.

Problem 4 — 7 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules for Problems 1 to 3, and in addition:

```
statements → statements statement
statement → PRINTLN LPAREN expr RPAREN SEMI
```

That is, you are to implement the println statement.

You may use the provided print.merl module to implement the println statement. **You must download this file from the link in the previous sentence and transfer it to your Linux environment account** – it is not accessed using `source /u/cs241/setup`. (If you are unable to access this file and you are enrolled in the course, contact the ISA at cs241@uwaterloo.ca.) The output from your compiler should now and henceforth be an assembly source file containing the line:

```
.import print
```

The `.import` directive lets you use a procedure from an external file via linking.

You can assemble such a file with `cs241.linkasm` instead of `cs241.binasm`. This will generate a MERL file. You should link this file with the library print.merl that we provide, using the command `cs241.linker`. This library contains a procedure labelled `print`, which outputs a decimal representation of the number passed to it in register `$1`. For example, you could assemble, link, and run the output of your code generator using the following commands:

```
cs241.linkasm < yourCompilersAssemblyLanguageOutput.asm > output.merl
cs241.linker output.merl print.merl > linked.merl
cs241.merl 0 < linked.merl > final.mips
mips.twoints final.mips
```

The `cs241.merl` command converts the linked MERL file to plain MIPS code, stripping out the relocation and linking metadata and relocating the program to run at address 0. This is not really necessary right now, but the memory allocation code you will use in Assignment 10 does not work correctly if MERL metadata is present.

Click here to go back to the top of Assignment 9.

Problem 5 — 8 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle variable declarations and assignment statements. Your solution should handle WLP4 programs whose syntax conforms to the rules given in Problems 1 to 4, and in addition:

```
dcls → dcls dcl BECOMES NUM SEMI
statement → lvalue BECOMES expr SEMI
lvalue → ID
lvalue → LPAREN lvalue RPAREN
```

Click here to go back to the top of Assignment 9.

Problem 6 — 8 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Problems 1 to 5 and in addition:

```
test → expr LT expr
statement → WHILE LPAREN test RPAREN LBRACE statements RBRACE
```

That is, you are to implement while loops whose continuation condition is expressed with "<".

Click here to go back to the top of Assignment 9.

Problem 7 — 8 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Problems 1 to 6 and in addition:

```
test → expr EQ expr
test → expr NE expr
test → expr LE expr
test → expr GE expr
test → expr GT expr
```

That is, you are to implement the other five comparison relations.

Click here to go back to the top of Assignment 9.

Problem 8 — 8 marks of 60 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Problems 1 to 7 and in addition:

```
statement → IF LPAREN test RPAREN LBRACE statements RBRACE ELSE LBRACE statements RBRACE
```

This includes all WLP4 programs that do not involve pointers or procedures.

Click here to go back to the top of Assignment 9.

CS 241 — Spring 2020 — Assignment 10

↑ Assignment 9

Wednesday, July 29th, at 5:00 pm

Assignments for CS 241

Assignment 10

Wednesday, August 5th, at 11:59 pm

P1 • P2 • P3 • P4 • P5 • P6 • Bonus

Problem 1 — 15 marks of 105 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Assignment 9 and in addition:

```
type → INT STAR
dcls → dcls dcl BECOMES NULL SEMI
factor → NULL
factor → AMP lvalue
factor → STAR factor
lvalue → STAR factor
```

For this problem and the following problem, we will not be doing any pointer comparisons other than equality (`==`) or inequality (`!=`).

This problem includes WLP4 programs that involve pointers, but do not do dynamic memory allocation, pointer arithmetic, or pointer comparison. Although the C++ standard leaves the result of dereferencing a null pointer undefined, we require that dereferencing NULL must crash the MIPS machine.

Click here to go back to the top of Assignment 10.

Problem 2 — 15 marks of 105 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Assignment 9, Assignment 10 Problem 1, and in addition:

```
factor → NEW INT LBRACK expr RBRACK
statement → DELETE LBRACK RBRACK expr SEMI
```

You may continue to assume that the parse tree of the WLP4 program does not contain any of the production rules that were excluded in Problem 1.

This problem includes WLP4 programs that do dynamic memory allocation, but do not do pointer arithmetic or pointer comparison.

We provide a module alloc.merl that implements a memory allocator, which you should use to implement the new and delete statements. **You must download this file from the link in the previous sentence and transfer it to your Linux environment account** – it is not accessed using `source /u/cs241/setup`. (If you are unable to access this file and you are enrolled in the course, contact the ISA at cs241@uwaterloo.ca.) Note that `alloc.merl` must be linked *last*.

The memory allocator contains three MIPS procedures: `init`, `new`, and `delete`. You must import each procedure individually (by writing `.import init`, `.import new`, and `.import delete` on three separate lines, rather than simply `.import alloc`).

The `init` procedure must be called once at the beginning of your generated program before any calls to `new` or `delete`.

- If the first parameter to `wain` is of type `int*` (i.e., the generated code will be passed an array), then the size of this array must be in register `$2` when `init` is called. Note that `mips.array` already puts the size of the array in register `$2`, so you only need to make sure that your generated code does not change `$2` before it calls `init`.
- If the first parameter to `wain` is of type `int`, then register `$2` must contain the value 0 (zero) when `init` is called.

The `new` procedure produces in register `$3` the address of the beginning of a contiguous sequence of *n* words of memory, where *n* is the value passed in register `$1` when the procedure was called. If `new` fails in the allocation, it will return `0`. Your generated code should return NULL if `new` fails, rather than `0`, since `0` is a valid memory address.

The `delete` procedure frees for reuse the block of memory whose address is passed in register `$1` when the procedure is called. The address passed to `delete` must be an address that was earlier returned from `new`, and has

not since already been passed to `delete`. You do not need to enforce this at compile time; it is the programmer's job to make sure that they use delete properly. However, note that passing NULL to `delete` is valid in C++ and the result is that nothing happens (no call to `delete` is made); your generated code should mimic this behaviour.

Marmoset uses a modified version of this memory allocator that prints debugging information to verify that you are calling init, new and delete correctly. Thus even if your program produces the correct output, it may not pass the Marmoset tests if you are not calling init, new and delete at the correct points with the correct argument values.

From now on, remember to use the `cs241.merl` command to remove MERL metadata from your program before executing it. Otherwise, calling `init` may overwrite part of the input array when used with `mips.array`, if the input array is sufficiently large. This is how you should assemble and link your compiler output from now on:

```
cs241.linkasm < yourCompilersAssemblyLanguageOutput.asm > output.merl
cs241.linker output.merl print.merl alloc.merl > linked.merl
cs241.merl 0 < linked.merl > final.mips
```

Click here to go back to the top of Assignment 10.

Problem 3 — 15 marks of 105 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Assignment 9, Assignment 10 Problems 1 and 2, and in addition:

```
expr → expr PLUS term
expr → expr MINUS term
```

This problem includes WLP4 programs that do pointer arithmetic but do not do pointer comparison.

Click here to go back to the top of Assignment 10.

Problem 4 — 15 marks of 105 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Assignment 9, Assignment 10 Problems 1-3, and in addition:

```
statement → IF LPAREN test RPAREN LBRACE statements RBRACE ELSE LBRACE statements RBRACE
statement → WHILE LPAREN test RPAREN LBRACE statements RBRACE
test → expr EQ expr
test → expr NE expr
test → expr LT expr
test → expr LE expr
test → expr GE expr
test → expr GT expr
```

This problem includes all WLP4 programs consisting of only the main procedure `wain`.

Click here to go back to the top of Assignment 10.

Problem 5 — 15 marks out of 105 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle WLP4 programs whose syntax is described by the production rules in Assignment 9, Assignment 10 Problems 1-4, and in addition:

```
procedures → procedure procedures
procedure → INT ID LPAREN params RPAREN LBRACE dcls statements RETURN expr SEMI RBRACE
params →
factor → ID LPAREN RPAREN
```

This problem includes all WLP4 programs in which all procedures other than `wain` take no arguments.

Click here to go back to the top of Assignment 10.

Problem 6 — 30 marks out of 105 (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Extend your WLP4 compiler to handle the full WLP4 language. All that remains is adding support for procedures with a non-zero number of parameters.

Click here to go back to the top of Assignment 10.

Bonus Problem — 6% bonus on Assignment 10 grade (filename: `wlp4gen.rkt` or `wlp4gen.cc`)

Modify your WLP4 compiler to reduce the size of the MIPS program that it generates for a particular WLP4 program (that is, the size in bytes of the binary MIPS program after it is assembled, not the number of lines or number of bytes in the assembly code that your compiler outputs). Although there exists a minimal MIPS program for a given WLP4 program, determining such a program is in general an undecidable problem.

Submissions that generate a MIPS program which is less than **180 000** bytes will get the 6% bonus.

A scoreboard of everyone's best submission for the bonus is available.

The list is anonymized but if you choose to enter, you can see how you are doing compared to others in the class.

Click here to go back to the top of Assignment 10.