C-net Programming Language Proposal

Rediet Bekele - rsb2179	- Manager
Kidus Mulu - km3533	- Language Guru
William Oseghare -who2103	- Systems Architecture
Bruk Zewdie - bbz2103	- Tester

Motivation

Our team wants to design a C-like language that is easy to use for file and network I/O. The language, named C-net, will contain a subset of C functionality coupled with additional elements that will allow for simpler network/file programming through succinct code and easy programmer interaction with file and network I/O. In addition to this, C-net incorporates a simplistic framework for multithreading to allow for use-cases such as listening for connections on different threads. In the process, we wanted to discard C's semantics for dynamic memory allocation and have it so that most program data goes on the heap by default like in Java.

Overview

C-net is a programming language that is founded on the C programming language. The basic syntax of C-net (declarations, loops, conditionals etc.) will be very similar to C as can be seen in the example code section. C-net is an imperative, specifically a procedural, language. Similar to C, C-net will be statically typed. In addition to these features, C-net aims to incorporate the following elements

1. **Easier network/file programming**

C-net provides an abstract wrapper for network sockets and files as objects for reading and writing along with built-in methods for performing common manipulations. In doing so, the language simplifies I/O for succinct and clear code. The goal is to present files and sockets as very similar things that can be written to and read from. To this end, the operations on File and Socket objects will be similar as much as possible.

These two objects, File and Socket, will be the hallmarks of the language. They will provide a simple interface for reading and writing data, and all buffering and memory management that is related to the operations is handled internally.

If the user wants to read a line of text from a socket, for example, all the user has to specify is the maximum number of characters they would like to read. The language then allocates the buffer, reads from the network socket until the end of the line, gives the data to the user and deallocates the buffer automatically.

The programming language will natively provide common operations such as reading and writing lines of text, reading a certain number of bytes or transferring data from one IO object to another.

2. **Multithreaded socket listening**

C-net will have a built-in function called **new_thread** that takes a listening socket and a user-defined function. The built-in function will then create a new thread and call the user-defined function every time a new connection is made. This will allow servers to handle multiple clients at a time.

3. **Dynamic memory allocation**

C-net provides heap memory allocation in the same way as Java: storing the data on the heap and the reference on the stack. Except for these references and primitive types, all user data will be stored on the heap. The dereference (*) operator will not be needed and any operation on a variable will automatically dereference before accessing the data. For flexible memory allocation, the user will be able to declare structs in the same fashion as in C.

Memory allocation will be done using the “new” keyword and users will have to explicitly “delete” any variables allocated with “new” since the language does not have automatic garbage collection for user-defined types. The only exception to this is the String type (see below). Since the language has strong type-checking, any memory allocated will be according to the size of the data type that is being declared.

4. **Automatically managed String type**

C-net will provide a built-in String type, which is immutable and destroyed any time it goes out of scope or a reference to it is lost. For example, if the user sets a String variable to something else, the old String is immediately deleted. Assignment of one string to another will perform a deep copy so that the new variable has its own underlying character array. Like the majority of data types, Strings will be stored on the heap.

5. **Functions as arguments**

To allow the user to transform data as it is read from a file or a network socket, the language will allow passing functions as arguments to other functions (similar to function pointers in C). However, since there is no casting and the language will be strongly typed, the function being passed will have to match the exact signature that the receiving function is expecting.

Data Types

Data Type	Description	Size
char	Primitive Character type	1 byte
int	Primitive numeric type	4 byte
float	Primitive numeric type	4 byte
String	String literal type	Size of the struct which contains the char pointer and other metadata (e.g. length)
Socket	Wrapper for network IO	Dynamic
File	Wrapper for file IO	Dynamic
type[]	Array of one of the above types	Dynamic

Keywords

Keyword	Description
for	Control flow iterative loop
while	Iterative loop with condition
if/ else	Basic control flow condition
return	Return expression
new	Allocates memory on the heap
delete	Deallocate memory that has been allocated
continue	Skip over to next iteration of the loop
break	Break out of the loop

Operators

Operator	Data types that can use operator	Description of operation
+ - * /	int, float	Carries out arithmetic operations between operands
*	String	Similar to Python, * allows for replication of string.
=	Used as needed	Assignment operator
==, !=	char, int, float, string	Equality operator (deep equality for string)
&&, ||	Used as needed	Logical operations (and, or)
[ ]	Array, string	Index operator
//	Used as needed	Single line comment
/* */	Used as needed	Multi-line comment

Sample code

Example 1: GCD algorithm

```
int gcd(int a,int b){

      while(a!=b) {
                  if(a>b)
                        a -= b;
                  else
                        b -= a;
      }

      return a;
}
```

Example 2: A web server which can handle multiple clients at the same time: The protocol is that the client sends the name of the file it wants and the server replies with the file.

```
void handle_listen(Socket socket)
{
     String filename = socket.read_line(100);

     File requested_file = fopen(READ, filename);

     socket.writeall(requested_file.readall());

     exit(1);
}
int main()
{
     Socket listener = nopen(LISTEN, 80, TCP);
     Socket connected_sock;

     for (;;) {
                connected_sock = listener.wait_until_connection();
//blocks here
                new_thread(handle_listen, connected_sock);
     }
}
```

Example 3: copy one file into another while capitalizing it

```
int main()
{
      String source = "test.txt";
      String destination = "test_copy.txt";

      File sourcef = fopen(READ, source);
      File destf = fopen(WRITE, destination);

      transform_and_copy(sourcef, destf, transformation_function);

      return 0;
}

int transform_and_copy(
                File sourcef,
                File destf,
                String transformation_func(String s))
{
      String tmp = sourcef.read_line(100);

      while(tmp.length() > 0)
      {
                destf.write(transformation_func(tmp));
                tmp = sourcef.read_line(100);
      }
}

String transformation_function(String s)
{
      return s.upper();
}
```

Example 4: Simple chat server (equivalent C-program (server_old.c))

```
int main(String[] args){
      if(args.length() < 2) {
                stderr.print_line("No port provided");
                exit(1);
      }

      portno = atoi(args[1]);
      Socket listener = fopen(LISTEN, portno, TCP);
      Socket connected_sock = listener.wait_until_connection(); // blocks here

      if (connected_sock.ERR > 0)
                error("error accepting new connection");

      while(1) {
                String message = connected_sock.read_line(255);
                if (message.length() == 0)
                      stderr.print_line("Error reading from client");

                stdout.print("Client: ");
                stdout.print_line(message);
                if (message == "Bye")
                      break;

                stdout.print_line("Server: ");
                message = stdin.read_line(255);

                connected_sock.write(message);
                if (connected_sock.ERR != 0)
                      error("Error writing to server\n")
      }

      connected_sock.close();
      listener.close();
      return 0;
}
```

Example 5: Memory allocation and struct declaration

```
struct person {
      String name;
      int age;
};

int main(String[] args)
{
      struct person p1 = new struct person;
      p1.name = "Joe";
      p1.age = 40;

      stdout.print_line("What is your name: ")
      p1.name = stdin.read_line(); //the old String is destroyed here

      stdout.print_line("How old are you?");
      p1.age = atoi(stdin.read_line());

      stdout.print_line("Print to file?");
      if(stdin.read_line() == "yes")
      {
      File f = fopen(WRITE, p1.name);
      f.write("Name: " + p1.name + "\n\n" + "age: " + p1.age);
      }

      delete p1;

      return 0;
}
```