From Graphical Loop Invariants to Formal Verification

Auteur : Grosjean, Antoine
Promoteur(s) : Fontaine, Pascal; Donnet, Benoît
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil en informatique, à finalité spécialisée en "management"
Année académique : 2024-2025
URI/URL : http://hdl.handle.net/2268.2/23376

Getting Started

hello.c

```c
#include <stdio.h>

int main(void) {
  printf("Hello World!\n");

  return 0;
}
```

Compile `hello.c` file with `gcc`

```
$ gcc hello.c -o hello
```

Run the compiled binary `hello`

```
$ ./hello
```

Output => Hello World!

Variables

```c
int myNum = 15;

int myNum2; // do not assign, ther
myNum2 = 15;

int myNum3 = 15; // myNum3 is 15
myNum3 = 10;     // myNum3 is now

float myFloat = 5.99; // floating
char myLetter = 'D';  // character

int x = 5;
int y = 6;
int sum = x + y; // add variables

// declare multiple variables
int x = 5, y = 6, z = 50;
```

Constants

```c
const int minutesPerHour = 60;
const float PI = 3.14;
```

Best Practices

```c
const int BIRTHYEAR = 1980;
```

Comment

```c
// this is a comment
printf("Hello World!"); // Can con

/*Multi-line comment, print Hello
to the screen, it's awesome */
```

Switch

```c
int day = 4;

switch (day) {
  case 3: printf("Wednesday"); bre
  case 4: printf("Thursday"); brea
  default:
    printf("Weekend!");
}
// output -> "Thursday" (day 4)
```

While Loop

```c
int i = 0;

while (i < 5) {
  printf("%d\n", i);
  i++;
}
```

NOTE: Don't forget to increment the variable used in the condition, otherwise the loop will never end and become an "infinite loop"!

Condition

```c
int time = 20;
if (time < 18) {
  printf("Goodbye!");
} else {
  printf("Good evening!");
}
// Output -> "Good evening!"
int time = 22;
if (time < 10) {
  printf("Good morning!");
} else if (time < 20) {
  printf("Goodbye!");
} else {
  printf("Good evening!");
}
// Output -> "Good evening!"
```

For Loop

```c
for (int i = 0; i < 5; i++) {
  printf("%d\n", i);
}
```

Break out of the loop Break/Continue

```c
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);
}
```

break out of the loop when `i` is equal to `4`

```c
for (int i = 0; i < 10; i++) {
  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

Example to skip the value of `4`

Do/While Loop

```c
int i = 0;

do {
  printf("%d\n", i);
  i++;
} while (i < 5);
```

While continue example

```c
int i = 0;

while (i < 10) {
  i++;

  if (i == 4) {
    continue;
  }
  printf("%d\n", i);
}
```

While Break Example

```c
int i = 0;

while (i < 10) {
  if (i == 4) {
    break;
  }
  printf("%d\n", i);

  i++;
}
```

Strings

```c
char greetings[] = "Hello World!";
printf("%s", greetings);
```

access string

```c
char greetings[] = "Hello World!";
printf("%c", greetings[0]);
```

modify string

```c
char greetings[] = "Hello World!";
greetings[0] = 'J';

printf("%s", greetings);
// prints "Jello World!"
```

Another way to create a string

```c
char greetings[] = {'H','e','l','l
printf("%s", greetings);
// print "Hell!"
```

Creating String using character pointer (String Literals)

```c
char *greetings = "Hello";
printf("%s", greetings);
// print "Hello!"
```

NOTE: String literals might be stored in read-only section of memory. Modifying a string literal invokes undefined behavior. You can't modify it.!

C **does not** have a String type, use `char` type and create an `array` of characters

Arrays

```c
int myNumbers[] = {25, 50, 75, 10
printf("%d", myNumbers[0]);
// output 25
```

change array elements

```c
int myNumbers[] = {25, 50, 75, 10
myNumbers[0] = 33;

printf("%d", myNumbers[0]);
```

Loop through the array

```c
int myNumbers[] = {25, 50, 75, 10
int i;

for (i = 0; i < 4; i++) {
  printf("%d\n", myNumbers[i]);
}
```

set array size

```c
// Declare an array of four intege
int myNumbers[4];

// add element
myNumbers[0] = 25;
myNumbers[1] = 50;
myNumbers[2] = 75;
myNumbers[3] = 100;
```

Operators

Arithmetic Operators

```c
int myNum = 100 + 50;
int sum1 = 100 + 50; // 150 (100 +
int sum2 = sum1 + 250; // 400 (150
int sum3 = sum2 + sum2; // 800 (40
```

+	Add	x + y
-	Subtract	x - y
*	Multiply	x * y
/	Divide	x / y
%	Modulo	x % y
++	Increment	++x
--	Decrement	--x

Assignment operator

x = 5	x = 5
x += 3	x = x + 3
x -= 3	x = x - 3
x *= 3	x = x * 3
x /= 3	x = x / 3
x %= 3	x = x % 3
x &= 3	x = x & 3
x |= 3	x = x | 3
x ^= 3	x = x ^ 3
x >>= 3	x = x >> 3
x <<= 3	x = x << 3

Comparison Operators

```c
int x = 5;
int y = 3;

printf("%d", x > y);
// returns 1 (true) because 5 is g
```

==	equals	x == y
!=	not equal to	x != y
>	greater than	x > y
<	less than	x < y
>=	greater than or equal to	x >= y
<=	less than or equal to	x <= y

Comparison operators are used to compare two values

Logical Operators

&&	and logical	returns true if both statements are true	`x < 5 && x < 10`
||	or logical	returns true if one of the statements is true	`x < 5 || x < 4`
!	not logical	Invert result, return false if true	`!(x < 5 && x < 10)`

Bitwise operators

&	Bitwise AND operation, "AND" operation by binary digits	`(A & B)` will get `12` which is 0000 1100
|	Bitwise OR operator, "or" operation by binary digit	`(A | B)` will get `61` which is 0011 1101
^	XOR operator, perform "XOR" operation by binary digits	`(A ^ B)` will get `49` which is 0011 0001
~	Inversion operator, perform "inversion" operation by binary bit	`(~A)` will get `-61` which is 1100 0011
<<	binary left shift operator	`A << 2` will get `240` which is 1111 0000
>>	binary right shift operator	`A >> 2` will get `15` which is 0000 1111

Operator Examples

```c
unsigned int a = 60; /*60 = 0011 1
unsigned int b = 13; /*13 = 0000 1
int c = 0;

c = a & b; /*12 = 0000 1100 */
printf("Line 1 -the value of c is

c = a | b; /*61 = 0011 1101 */
printf("Line 2 -the value of c is
c = a ^ b; /*49 = 0011 0001 */
printf("Line 3 -the value of c is
c = ~a; /*-61 = 1100 0011 */
printf("Line 4 -The value of c is
c = a << 2; /*240 = 1111 0000 */
printf("Line 5 -the value of c is
c = a >> 2; /*15 = 0000 1111 */
printf("Line 6 -The value of c is
```

Data Types

Basic data types

`char`	1 byte	`-128 ~ 127`	single character/alphanumeric/ASCII
`signed char`	1 byte	`-128 ~ 127`	-
`unsigned char`	1 byte	`0 ~ 255`	-
`int`	2 to 4 bytes	`-32,768 ~ 32,767`	store integers
`signed int`	2 bytes	`-32,768 ~ 32,767`	
`unsigned int`	2 bytes	`0 ~ 65,535`	
`short int`	2 bytes	`-32,768 ~ 32,767`	
`signed short int`	2 bytes	`-32,768 ~ 32,767`	
`unsigned short int`	2 bytes	`0 ~ 65,535`	
`long int`	4 bytes	`-2,147,483,648 ~ 2,147,483,647`	
`signed long int`	4 bytes	`-2,147,483,648 ~ 2,147,483,647`	
`unsigned long int`	4 bytes	`0 ~ 4,294,967,295`	
`float`	4 bytes	`3.4E-38 ~ 3.4E+38`	
`double`	8 bytes	`1.7E-308 ~ 1.7E+308`	
`long double`	10 bytes	`3.4E-4932 ~ 1.1E+4932`	

Data types

```c
// create variables
int myNum = 5; // integer
float myFloatNum = 5.99; // floati
char myLetter = 'D'; // string
// High precision floating point d
double myDouble = 3.2325467;
// print output variables
printf("%d\n", myNum);
printf("%f\n", myFloatNum);
printf("%c\n", myLetter);
printf("%lf\n", myDouble);
```

`char`	character type
`short`	short integer
`int`	integer type
`long`	long integer
`float`	single-precision floating-point type
`double`	double-precision floating-point type
`void`	no type