Virtual Lab Computer Science and Engineering and IT

IIIT Hyderabad: Computer Programming LAB

JUIT Lab: Computer Programming LAB

(10B17CI171)

1. Pointers
2. Expression Evaluation

---

1. Pointers

Introduction

Every time a variable is declared, some bytes are allocated to that variable depending on its datatype. For example, int takes 4 bytes, a char takes only one byte. Arrays allow allocation of even larger number of storage space or "bytes". Each of these bytes has a distinct address in the main memory. Pointer is a datatype which can store this address. Hence a pointer can serve as a "reference" to the data contained other variables. This has some useful applications, most importantly, allowing users to modify data from within a function without needing to make a copy of the data, and also allowing the user to make "linked" data structures.

Theory

A pointer is a programming language data type which can store memory addresses of other variables. A pointer variable corresponding to any data type can be declared by using * before the name of the pointer.

```
int *pointer1_1,*pointer2_i, var_i=5;
```

This would declare 2 integer pointers meant to store references(memory address) to variables of integer data type. Note that var_i is just a normal integer variable. Alternatively, pointers can also be declared as:

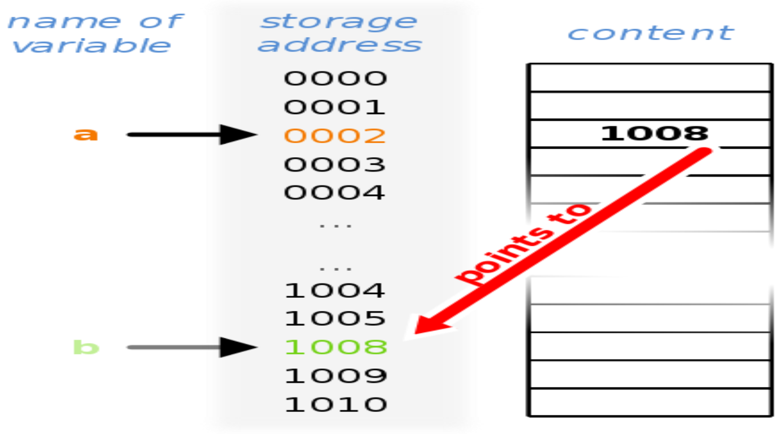
```
int* pointer1_1,pointer2_i;
int var_i=5;
```

The & operator can be used with any variable to get its memory address. So, writing

```
pointer1_i=&var_i;
```

would assign the memory address of var_i to pointer1_i.

Figure explaining the working of the statement: a=&b;



The * operator is used to access the values stored at a given address. So, writing

```
        printf("%d\n",*pointer1_i);
```

would print 5 on the console. This process of accessing the value stored at a given position is known as dereferencing a pointer. Similarly, to store references to character or float variables one can define char or float type pointers.

A pointer can be used to allocate memory in the runtime using the malloc() function, defined in stdlib.h, by doing dynamic memory allocation. Writing

```
        int *ptr=(int *)malloc(20*(sizeof(int));
```

will allocate a space 20 integer variables and store the address of the first byte in ptr. An array, infact, is just a constant pointer to which allocation is done automatically. Hence, writing

```
        int arr[100];
```

is equivalent to writing

```
        const int* arr=(int *)malloc(100*sizeof(int));
```

So, writing *arr will give the value stored at arr[0].

Arithmatical operations like addition and subtraction can be performed to a pointer. The nature of these arithmatic operations is what distinguishes an integer pointer from, say, a character pointer, which otherwise just store memory addresses for another variable. Adding one to an interger pointer makes to point to the next integer, and hence, it skips 4 bytes. Adding one to a character pointer makes it to point to the next character, and hence, it skips only 1 byte. Hence, for the array arr, writing arr[3] is equivalent to writing *(arr+3). Both refer to the value stored in the 4th cell of the array.

Objective

1.To understand the concept of memory adddress associated with every variable.
2.To understand pointer referencing and dereferencing.
3.To understand how pointers are useful in dynamic allocation of memory.

Manual

In this lab you will learn the basic concepts of pointers and addresses. You will also understand the process of memory allocation and passing in reference.

Procedure

Call By Value

1.Press start to start the experiment.
2.Press next to see the execution of the code.
3.Relavant line in the code is shown here.
4.The output of the code is shown in the right.

Call By Reference

1.Press start to start the experiment.
2.Press next to see the execution of the code.
3.The output of the code is shown in the right.
4.You can stop the code using stop button.

Further Reading

1.http://www.exforsys.com/tutorials/c-language/c-pointers.html
2.http://en.wikipedia.org/wiki/Pointer_(computer_science)

2. Expression Evaluation

Introduction

Evaluating an expression is one of the most common tasks in programming. For example a+b+c is an expression. Here + is an operator. The variables a, b and c are operands. The value of the expression is the sum of a, b and c. An expression may be often evaluated to assign a variable a new value or to verify the truthfulness of a test condition. For example the operator can be unary(Ex: ~,++ etc.), binary(Ex: +,* etc.) or ternary (Ex: (a>b)?p:q)

Examples of expressions:

1. To compute the area of a triangle from base b and height h
   Area=1/2.0*base*height
   Note that this expression will be evaluated left to right, that is the division opeartor will be evaluated, followed by two multiplication operators. The 2.0 is used to make sure that the division is carried out as a floating point operation.
2. To compute the area of a triangle from the three sides, a, b and c
   s=(a+b+c)/2.0
   area = s*(s-a)*(s-b)*(s-c)
   Here area is calculated in two steps. The first step, s is computed from the three sides a, b and c. Note that, the paranthesis is used so that evaluation within the paranthesis takes first. Without paranthesis, we would have obtained the value of s as sum of a, b and half of c. Similarly, paranthesis are very important in evaluating the second expression.
3. To find the nth term of an arithmetic progression, we can use the following expression
   tn= a+(n-1)*d
   where a is the first term and d is the common difference.
   Here also, the use of paranthesis has helped us in evaluating the expression the way we need it.

An expression used in a computer program is very similar to an algebraic expression and when many operators are specified in an expression, then the evaluation takes place according to the precedence of operators.The order of the precedence is given by a table of precedence.

A paranthesis can be used to override the precedence of operators and force the evalutation of a sub-expression within an expression. Expression is evaluated according to assosciativity of the operator. The assosciativity of the operator is a property that determines how operators of the same precedence are grouped in the absence of paranthesis. The objective in evaluating an expression is to consider the precedence and aasosciativity of the individual operators and compute the value of the sub-expressions.

Theory

Evaluating an expression involves repeatedly solving the sub-expressions in the lowest level paranthesis and substituting its value to solve the bigger expression. If an expression or sub-expression does not have a paranthesis, then we can directly compute its value based on the precedence and assosciativity of the operators present in the expression. That is, the sub expression involving operator with higher precedence is evaluated before other sub-expressions. The following table gives the precedence order of all the operators:

Table of precedence (Click here)

Operators can be broadly categorized as:

1. Arithmetic Operators: These operators are used for arithmetic operations like addition,subtraction and multiplication. Ex: DIVISION(/),MODULUS OR REMAINDER (%)
2. Relational Operators: These operators are typically used for comparison of two operands.Ex: GREATER THAN(>),GREATER THAN OR EQUAL TO(>=) etc.
3. Logical Operators: These operators are used for conducting logical operations like AND,OR. Ex: LOGICAL AND(&&),LOGICAL OR (||) etc.
4. Bitwise Operators: These operators are used to perform operations on bits. Ex: BITWISE AND(&), BITWISE OR (|) etc.

Objective

1. To learn about different types of operators.
2. To learn about the precedence of the operators.
3. To learn how to evaluate an expression.

Manual

Here we shall see the step by step evaluation of mathematical expressions.

Procedure

Procedure for the experiment is as follows.

1. Select the type of operators and the datatype you want to work upon from the top most bar.
2. You can edit the values of variables by pressing the edit button.
3. Select an expression prototype from the menu.
4. You can also edit this expression.
5. You can also edit this expression.
6. Press Next to see the step by step evaluation of the selected expression in the central panel and the corresponding reasoning in the right panel.
7. Press stop if you want to abort the experiment and start over.

Further Reading

1. http://en.wikipedia.org/wiki/Expression_(programming)
2. http://www.exforsys.com/tutorials/c-language/c-expressions.html
3. Chapter 2, The C Programming Language, Kernighan and Ritchie
4. Chapter 3, Programming with C, Brian Gottfried