Stack and Heap Memory

Differences between stack and heap memory:

<table>
<thead>
<tr>
<th>Stack</th>
<th>Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managed automatically by the computer. Applications do not need to concern themselves with stack memory management.</td>
<td>Not managed automatically. Application programmer is responsible for allocation and de-allocation.</td>
</tr>
<tr>
<td>Very fast access.</td>
<td>Relatively slower access.</td>
</tr>
<tr>
<td>Memory does not become fragmented as the allocation and de-allocation is LIFO.</td>
<td>No guaranteed efficient use of space. Memory may become fragmented over time as blocks are allocation, and freed.</td>
</tr>
<tr>
<td>Stack size is limited. Variables cannot be arbitrarily large.</td>
<td>Heap allocation size can be as large as the computer's memory.</td>
</tr>
<tr>
<td>Example: Declaration of any local variables in functions in C++.</td>
<td>Example: the use of new keyword in C++ or Java.</td>
</tr>
</tbody>
</table>

Code demo and example:

```cpp
int foo()
{
    int a, b;       // space for a and b is allocated in stack
    int *p;         // space for p is allocated in stack
    p = new int;    // space for the new variable is allocation in
                    // heap, and its address is stored in p
    return 0;
}
```

Why and when do we use heaps?

1. The amount of memory needed by the program is too large. In this case, heaps are used since stack has a restriction on the amount of memory that can be taken up by a program/function.
2. We do not know the amount of data that is going to be used by the program. For example, if the number of inputs are undetermined.

Pointers in C++

Every variable that we declare in C++ or any other programming language is stored somewhere in memory. Each memory location has an address.

Variables typically hold values. For example, a variable of type int holds an integer values, a variable of type char holds a character, and a variable of type float holds a floating point decimal number. A pointer
is a special kind of variable which can hold the address of another variable. In other words, a pointer is a variable whose value is the address of another variable which may be holding an integer, character, floating point or any other value.

Example:

```c++
int a = 10;
int* p;
p = &a;
```

In the above example, p is a pointer to an integer type of variable, or p holds the address of a variable which, in turn, holds an integer value.

What happens if we try to print out the value of p in the above example?

```c++
cout << p;
```

Output will be something like 0x723956, basically something of the format 0x<some hexadecimal number>. This value is the address of variable ‘a’.

Some basic operators to know about:

1. **Reference operator (&):** This operator returns the address of the variable that follows it. Example: &a returns the address of the variable a.
2. **Dereference operator (*):** This operator accesses the value at the address that is stored in a pointer variable. Example: If p is a pointer to variable a, then *p returns the value that is stored in variable a.

When do we use pointers?

Implementation of linked lists, or in general, whenever we need to dynamically allocate memory.

**Dynamic Memory Allocation and Deletion in C++**

We use the keyword new in C++ to allocate memory dynamically.

Example:

```c++
int *p = new int;
```

The new keyword allocates a certain amount of space in the heap and returns the address to this memory location. If we want to be able to do anything with this allocated memory location, we need to capture this returned address otherwise it’ll be lost. We do this with the help of a pointer variable.

When we are done using a variable and when a variable (that was dynamically allocated) is no longer required, we free its memory space by using the delete keyword.

Example:

```c++
delete p;
```
Exercises:

Exercises 1. from the book:

If i is an integer and p and q are pointers to integers, which of the following assignments cause a compilation error?

Solutions:

1. p = &i;  Compiles, p holds the address of i
2. p = *&i;  Doesn’t compile
3. p = &*i;  Doesn’t compile
4. i = **&p  Compiles, i holds a value equal to *p
5. i = *&p;  Doesn’t compile
6. i = &*p;  Doesn’t compile
7. p = &*&i;  Compiles, p holds address of i
8. q = *&&p;  Doesn’t compile
9. q = **&p;  Doesn’t compile
10. q = *&p;  Compiles, q and p point to the same int variable
11. q = &*p;  Compiles, same as 10 above

Exercise 2:

Will the following code compile? If yes, What will be the output of the following piece of code?

```cpp
#include<iostream>
using namespace std;

int main()
{
    int x = 10;
    int *p = new int(30);
    int **q, *r;
    r = &x;
    q = &*p;
    *q = r;
    cout << *p << endl << **q << endl << *r;
}
```