



---

# MEMORY ALLOCATOR

---

OS assignment CSE 15



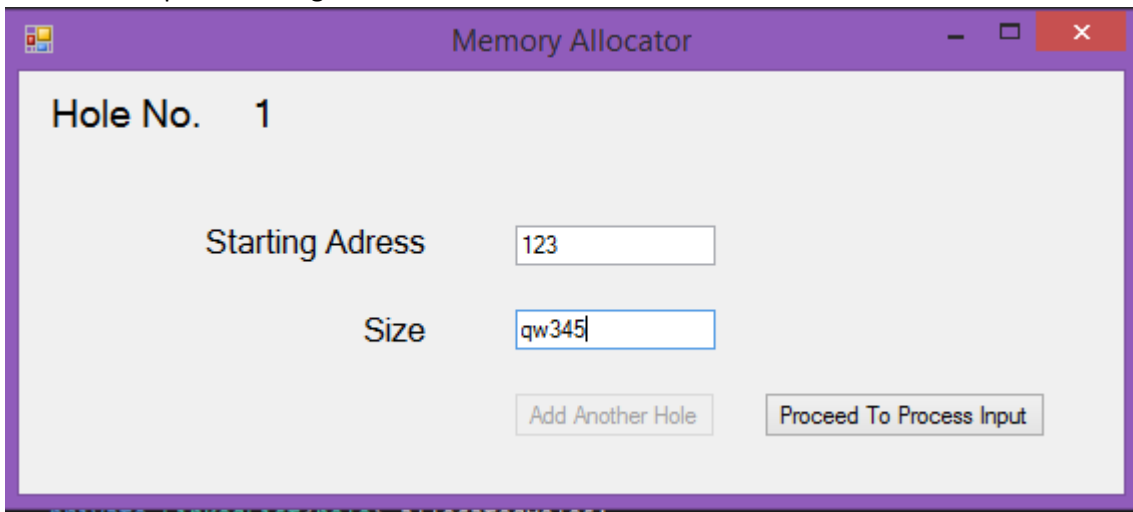
SUBMITTED BY:AMR MOAHMED AHMED MAHMOUD, SEC: 2

SUBMITTED TO: ENG/ MAY MOHAMED

# Manual

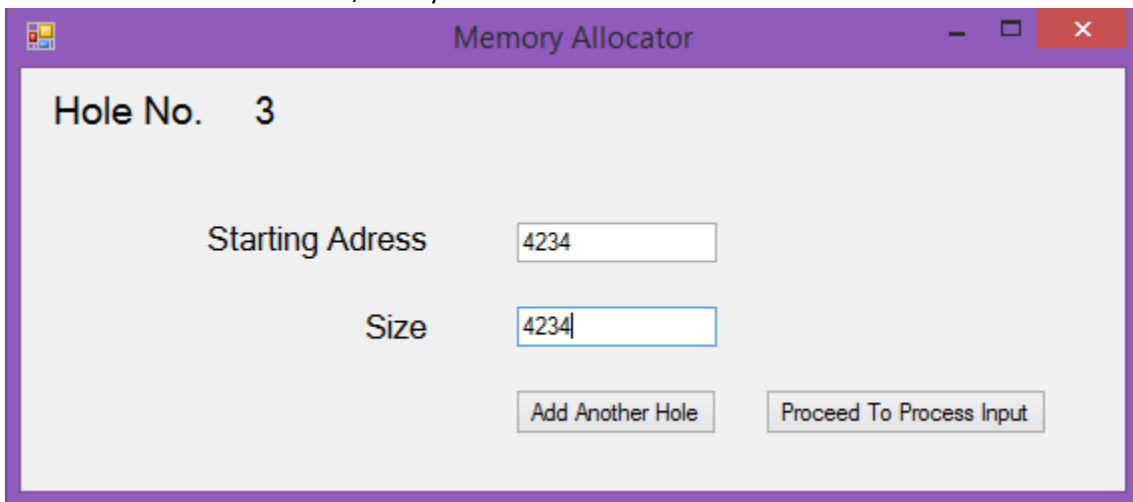
## 1. Entering the holes

- Open the .exe file
- Enter a holes' start address and size
- Note that the button "Add Another Hole" will always be disabled until both start and size field contain valid positive integer numbers



The screenshot shows a window titled "Memory Allocator" with a purple title bar. Inside, it displays "Hole No. 1". Below this, there are two input fields: "Starting Address" with the value "123" and "Size" with the value "qw345". At the bottom, there are two buttons: "Add Another Hole" and "Proceed To Process Input". The "Add Another Hole" button is disabled (grayed out), while the "Proceed To Process Input" button is active.

- When you are done entering the hole (note that you can enter as many hole as you want) press "Process to process input" to start entering processes; note that "proceed to process input also reads the start and size fields, if they contain valid values a hole will be added with these values



The screenshot shows the same "Memory Allocator" window, but now it displays "Hole No. 3". The "Starting Address" field contains "4234" and the "Size" field contains "4234". The "Add Another Hole" button remains disabled, and the "Proceed To Process Input" button remains active.

- Start entering processes' sizes
- Note that also the button "Add Another Process" will be disabled until the size field contain a valid positive integer only

The screenshot shows a window titled "Memory Allocator" with a purple header bar. Inside, the text "Process No. 1" is at the top. Below it, the label "Process Size" is followed by a text input field containing the value "4343". At the bottom, there are two buttons: "Add Another Process" and "Choose Allocator Type".

- Enter as many process as you want(no limit on process numbers) then press “choose allocator type ” button; also note that this button reads the process size field and if it contain a valid int a process will be added of that size

The screenshot shows the same "Memory Allocator" window, but now for "Process No. 9". The "Process Size" input field now contains the value "324". The "Add Another Process" and "Choose Allocator Type" buttons remain at the bottom.

- Choose the type of allocator you want, then press “Start Allocation” button

The screenshot shows the "Memory Allocator" window with the title "Choose An Allocator". It features three radio button options: "Best Fit" (which is selected), "Worst Fit", and "First Fit". A "Start Allocation" button is located at the bottom right of the window.

- The following screenshot was taken after the following test case was entered allocator type "Best Fit"

Hole number	Start	Size
1	0	10
2	6	50
3	40	300
4	500	1000
5	4500	2300
6	10000	300

Process number	Size
1	10
2	30
3	50
4	100
5	900
6	5000
7	200

The screenshot shows a memory allocation simulation window titled "output". It contains three main panels:

- UnAllocated Processes:** A list of processes P1 through P7 with their respective sizes:
 

P1	Size: 10
P2	Size: 30
P3	Size: 50
P4	Size: 100
P5	Size: 900
P6	Size: 5000
P7	Size: 200
- UnAllocated Holes:** A list of five hole blocks with their start, size, and end addresses:
 

Start: 40	Size: 300	End: 340
Start: 500	Size: 1000	End: 1500
Start: 4500	Size: 2300	End: 6800
Start: 10000	Size: 300	End: 10300
Start: 0	Size: 56	End: 56
- Allocated Holes:** This panel is currently empty.

On the right side of the window, there are two buttons: "Allocate Next Step" and "Allocate All".

- Note that the holes that can be merged are automatically merged at every step of the program

- Now we have 2 choices “Allocate All” and “Allocate next Step”  
 “Allocate all” : view final state of hole and processes and disable the 2 buttons  
 “Allocate next step”: view state after allocation of one process and disable the 2 button when there are no processes left
- What follows is the output from the allocate all button

UnAllocated Processes	UnAllocated Holes	Allocated Holes
P6 Size: 5000	Start: 4500 Size: 2300 End: 6800	Start: 0 Size: 10, Process: P1 End: 10
	Start: 10150 Size: 150 End: 10300	Start: 10 Size: 30, Process: P2 End: 40
	Start: 1400 Size: 100 End: 1500	Start: 10000 Size: 50, Process: P3 End: 10050
	Start: 240 Size: 100 End: 340	Start: 10050 Size: 100, Process: P4 End: 10150
		Start: 500 Size: 900, Process: P5 End: 1400
		Start: 40 Size: 200, Process: P7 End: 240

- Process p6 can't be allocated as no hole is that big
- To enter another simulation you have to restart the app.

# Classes

## hole

Data
<pre>private int start; private int size; string process;</pre>
<pre>public hole() public hole(int x, int y) public hole(int x, int y, string p) public int getStart() public int getSize () public void setSize(int s) public void setStart(int s) public String getProcess() public void setProcess(string p)</pre>

## Process

Data
<pre>private int order; private int size; private bool allocated</pre>
<pre>public process() public process(int x, int y) public process (int x, int y, bool a) public int getOrder() public int getSize() public void setOrder(int x) public void setSize(int x) public void setAsAllocated() public bool isAllocated()</pre>

## allocator

Data
<pre>private int order; private int size; private bool allocated</pre>
<pre>public allocator() {     public void addHole(hole h)     public void addProcess(process p)     public int getNumberOfHoles()     public int getNumberOfProcesses()</pre>

```
public void refreshHoles(LinkedList <hole>h)
public LinkedList<hole> getHoles()
public LinkedList<process> getProcesses()
public LinkedList<hole> getAllocatedHoles()
public LinkedList<process> getUnAllocatedProcesses()
public void setType(string s)
public string getType() { return type; }
public void firstFitAllocateProcess(process p)
public void bestFitAllocateProcess(process p)
public void worstFitAllocateProcess(process p)
public void firstFit()
public void bestFit()
public void worstFit()
```

## Variables

Data
<pre>public static allocator alloc;</pre>

# Code Snippets

```
Allocator::public void bestFitAllocateProcess(process p)
{
    if (variables.alloc.getHoles().Count() == 0)
    {
        unallocatedprocesses.AddLast(p);
        processes.Remove(p);
    }
    if (variables.alloc.getHoles().Count() > 0) this.refreshHoles(holes);
    hole bestFit = holes.ElementAt(0);
    bool matchFound = false;
    int bestFitIndex = 0;
    for (int i = 0; i < holes.Count(); i++)
    {
        if (holes.ElementAt(i).getSize() >= p.getSize())
        {
            p.setAsAllocated();
            processes.Remove(p);
            if ((holes.ElementAt(i).getSize() < bestFit.getSize() &&
holes.ElementAt(i).getSize() >=
p.getSize())&&matchFound)||((holes.ElementAt(i).getSize()>=p.getSize())&&!matchFound))
            {
                bestFit = holes.ElementAt(i);
                bestFitIndex = i;
            }
            matchFound = true;
        }
    }
    if (matchFound)
    {
        if (bestFit.getSize() == p.getSize())
        {
            holes.ElementAt(bestFitIndex).setProcess("P" +
p.getOrder().ToString());
            allocatedHoles.AddLast(holes.ElementAt(bestFitIndex));
            holes.Remove(holes.ElementAt(bestFitIndex));
        }
        else if (bestFit.getSize()>p.getSize())
        {
            allocatedHoles.AddLast(new hole(bestFit.getStart(), p.getSize(),
"P" + p.getOrder().ToString()));
            hole temp=holes.ElementAt(bestFitIndex);
            holes.Remove(holes.ElementAt(bestFitIndex));
            holes.AddLast(new hole(bestFit.getStart() + p.getSize(),
bestFit.getSize() - p.getSize()));
        }
    }
    else
    {
        unallocatedprocesses.AddLast(p);
        processes.Remove(p);
    }
}
```



```

    }

    }
Allocator:: public void bestFit()
{
    if (variables.alloc.getHoles().Count() > 0) this.refreshHoles(holes);
    for (int i = 0; i < processes.Count(); i++)
    {
        bestFitAllocateProcess(processes.ElementAt(i));
        this.refreshHoles(holes);
        variables.alloc.bestFit();
        break;
    }
}

```

```

Allocator::public void firstFitAllocateProcess(process p)
{
    if (variables.alloc.getHoles().Count() == 0)
    {
        unallocatedprocesses.AddLast(p);
        processes.Remove(p);
    }
    if (variables.alloc.getHoles().Count() > 0) this.refreshHoles(holes);
    for (int i = 0; i < holes.Count(); i++)
    {
        if (holes.ElementAt(i).getSize() >= p.getSize())
        {
            p.setAsAllocated();
            processes.Remove(p);
            if (holes.ElementAt(i).getSize() == p.getSize())
            {
                allocatedHoles.AddLast(new hole(holes.ElementAt(i).getStart(),
holes.ElementAt(i).getSize(), "P " + p.getOrder().ToString()));
                holes.Remove(holes.ElementAt(i));
                break;
            }
            else
            {
                hole temp = holes.ElementAt(i);
                hole unallocatedhole = new hole(temp.getStart() + p.getSize(),
temp.getSize() - p.getSize());
                holes.Remove(holes.ElementAt(i));
                allocatedHoles.AddLast(new hole(temp.getStart(), p.getSize(),
"P " + p.getOrder().ToString()));
                break;
            }
        }
        else
        {
            unallocatedprocesses.AddLast(p);
            processes.Remove(p);
        }
    }
}

```

```

Allocator:: public void firstFit()
{
    if (variables.alloc.getHoles().Count() > 0) this.refreshHoles(holes);

    for (int i=0;i<processes.Count();i++)
    {
        firstFitAllocateProcess(processes.ElementAt(i));
        this.refreshHoles(holes);
        variables.alloc.firstFit();
        break;
    }
}

```

```

Allocator:: public void worstFitAllocateProcess(process p)
{
    if (variables.alloc.getHoles().Count() == 0)
    {
        unallocatedprocesses.AddLast(p);
        processes.Remove(p);
    }
    if (variables.alloc.getHoles().Count() > 0) this.refreshHoles(holes);
    hole worstFit = holes.ElementAt(0);
    bool matchFound = false;
    int worstFitIndex = 0;
    for (int i = 0; i < holes.Count(); i++)
    {
        if (holes.ElementAt(i).getSize() >= p.getSize())
        {
            p.setAsAllocated();
            processes.Remove(p);
            if ((holes.ElementAt(i).getSize() > worstFit.getSize() &&
holes.ElementAt(i).getSize() >= p.getSize() && matchFound) ||
(holes.ElementAt(i).getSize() >= p.getSize() && !matchFound))
            {
                worstFit = holes.ElementAt(i);
                worstFitIndex = i;
            }
            matchFound = true;
        }
    }
    if (matchFound)
    {
        if (worstFit.getSize() == p.getSize())
        {
            holes.ElementAt(worstFitIndex).setProcess("P" +
p.getOrder().ToString());
            allocatedHoles.AddLast(holes.ElementAt(worstFitIndex));
            holes.Remove(holes.ElementAt(worstFitIndex));
        }
        else if (worstFit.getSize() > p.getSize())
        {
            allocatedHoles.AddLast(new hole(worstFit.getStart(), p.getSize(),
"P" + p.getOrder().ToString()));
            hole temp = holes.ElementAt(worstFitIndex);

```

```

        holes.Remove(holes.ElementAt(worstFitIndex));
        holes.AddLast(new hole(worstFit.getStart() + p.getSize(),
worstFit.getSize() - p.getSize()));

    }

}
else
{
    unallocatedprocesses.AddLast(p);
    processes.Remove(p);
}

}
Allocator:: public void worstFit()
{
    if (variables.alloc.getHoles().Count() > 0) this.refreshHoles(holes);
    for (int i = 0; i < processes.Count(); i++)
    {
        worstFitAllocateProcess(processes.ElementAt(i));
        this.refreshHoles(holes);
        variables.alloc.worstFit();
        break;
    }
}
}

```

```

Allocator:: public void refreshHoles(LinkedList <hole>h)// do possible merges between
//holes
{
    for (int i = 0; i < h.Count(); i++)
    {
        for (int j = 0; j < h.Count(); j++)
        {
            if (i == j) ;
            else
            {
                if (h.ElementAt(i).getStart() + h.ElementAt(i).getSize() >=
h.ElementAt(j).getStart() && h.ElementAt(i).getStart() <= h.ElementAt(j).getStart())
                {
                    hole temp = new hole();
                    hole removable = h.ElementAt(j);// h.Remove(h.ElementAt((i
< j) ? j - 1 : j));
                    temp.setStart((h.ElementAt(i).getStart() <
h.ElementAt(j).getStart()) ? h.ElementAt(i).getStart() : h.ElementAt(j).getStart());
                    temp.setSize((h.ElementAt(i).getStart() +
h.ElementAt(i).getSize() > h.ElementAt(j).getStart() + h.ElementAt(j).getSize()) ?
h.ElementAt(i).getStart() + h.ElementAt(i).getSize() - temp.getStart() :
h.ElementAt(j).getStart() + h.ElementAt(j).getSize() - temp.getStart());
                    h.Remove(h.ElementAt(i));
                    h.Remove(removable);
                    h.AddLast(temp);
                    variables.alloc.refreshHoles(h);
                    break;
                }
            }
        }
    }
}

```

```
        } break;
    }
}
```