In the ANF version, Quiz-Allocating Pairs on the Heap

Example: Allocation

NOTE:

<table>
<thead>
<tr>
<th>1. Representation</th>
</tr>
</thead>
</table>

2. Increment

Let's use a Allocating Addresses

ANF will ensure that expressions like:

Next, let's look at how to implement pair 2. Construction

which require us to put data on the heap.

Instead, let's learn how to make

Pairs: Semantics (Behavior)

So actual addresses, written in binary, omitting trailing zeros, are of the form

A Problem: Numbers vs. Pointers?

Each of the above corresponds to a

Creating Heap Data Structures

Recall that we

Next, let's add support for

Implementing the above semantics. We need to work out strategies for:

1. Representation

2. Construction

Next, let's ponder what exactly we're trying to achieve.

Constructing

A pair

First, let's informally develop a strategy for extending our language with pairs,

For example,

How many words would we need to store the tuple

Recall that we

Single step…

While our

TNumber

Stack vs. Heap

Which is great because at each address, we have a pair, i.e. a

Single word

Pairs (e0, e1)

Creating Heap Data Structures

In the process of doing so, we will learn about

Using pointers.

Pointers

So actual addresses, written in binary, omitting trailing zeros, are of the form

2. Increment

Let's use a Allocating Addresses

ANF will ensure that expressions like:

Each of the above corresponds to a

Stack vs. Heap

Which is great because at each address, we have a pair, i.e. a

Single word

Pairs (e0, e1)

Creating Heap Data Structures

In the process of doing so, we will learn about

Using pointers.

Pointers

So actual addresses, written in binary, omitting trailing zeros, are of the form

2. Increment

Let's use a Allocating Addresses

ANF will ensure that expressions like:

Each of the above corresponds to a

Stack vs. Heap

Which is great because at each address, we have a pair, i.e. a

Single word

Pairs (e0, e1)

Creating Heap Data Structures

In the process of doing so, we will learn about

Using pointers.

Pointers

So actual addresses, written in binary, omitting trailing zeros, are of the form

2. Increment

Let's use a Allocating Addresses

ANF will ensure that expressions like:

Each of the above corresponds to a

Stack vs. Heap

Which is great because at each address, we have a pair, i.e. a

Single word

Pairs (e0, e1)

Creating Heap Data Structures

In the process of doing so, we will learn about

Using pointers.

Pointers

So actual addresses, written in binary, omitting trailing zeros, are of the form

2. Increment

Let's use a Allocating Addresses

ANF will ensure that expressions like:

Each of the above corresponds to a
Construct QUIZ

Initialize our code must take care of three things:

1. Accessing
2. Copying
3. Accessing

Let us extend our run-time (in c-bits/main.c)

As usual, let's continue with our recipe:

First, we set the tag bits of D.

Finally, we remove the tag bits by doing the opposite of

Pretty pictures are well and good, time to build stu

Here is a snapshot of the heap after the pair(s) are allocated.

Our code must take care of three things:

1. Accessing
2. Copying
3. Accessing

Finally, to

3. Accessing

First, we set the tag bits of D.

Finally, we remove the tag bits by doing the opposite of

Pretty pictures are well and good, time to build stu

Here is a snapshot of the heap after the pair(s) are allocated.

Our code must take care of three things:

1. Accessing
2. Copying
3. Accessing

Finally, to

3. Accessing

First, we set the tag bits of D.

Finally, we remove the tag bits by doing the opposite of

Pretty pictures are well and good, time to build stu

Here is a snapshot of the heap after the pair(s) are allocated.
We can now write various functions that build and operate on lists, for example, a function to sum up the elements of a list:

```
```