COMP1927

Generic ADTs in C

Function Pointers

- C can pass functions by passing a pointer to them.
- Function pointers ...
  - are references to memory addresses of functions
  - are pointer values and can be assigned/passed

- E.g. a pointer to a function mapping

  `int → int`

  `int (*fun)(int)`

- Function pointer variables/parameters are declared as:

  `typeOfReturnValue (*fname)(typeOfArguments)`

- Lectures slides on the topic "Function to Pointers" are drawn from the lecture slides prepared by Angela Finlayson (COMP1927 16x1)

Function Pointers

```
int square(int x) { return x*x;}

int timesTwo(int x) {return x*2;}

int (*fp)(int);

fp = &square;       //fp points to the square function

int n = (*fp)(10); //call the square function with input 10

fp = timesTwo;      //works without the &
                    //fp points to the timesTwo function

n = (*fp)(2);       //call the timesTwo function with input 2

n = fp(2);          //can also use normal function call
                    //notation
```

Higher-order Functions

- Functions that get other functions as arguments, or return functions as a result
- **Example:** the function traverse takes a list and a function pointer as argument and applies the function to all nodes in the list

```
void printList(link ls){
    link curr = ls;
    while(curr != NULL){
        printf("%d ",curr->data);  //Process the node
        curr = curr->next;
    }
}
//-------------------------------------------------
// apply function fp to all nodes in ls
void traverse (link ls, void (*fp) (link) ){
    link curr = ls;
    while(curr != NULL){
        (*fp) (curr);
        curr = curr->next;
    }
}
```

First parameter is **ls**, of type link

Second parameter is **fp**,
Pointer to a function like,
void functionName(link ls)

```
// To call the function, function
// must have matching prototype

traverse(myList, printList);
```

Higher-order Functions

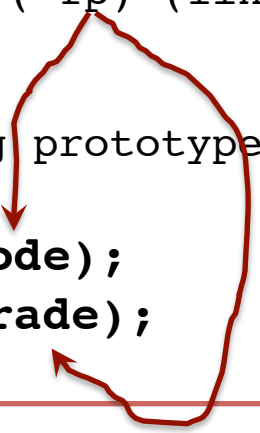
```
void printNode(link ls){
    if(ls != NULL){
        printf("%d->",ls->data);
    }
}
```

```
void printGrade(link ls){
    if(ls != NULL){
            if(ls->data>= 85){
            printf("HD ");
        }
        else {
            printf("FL ");
        }
    }
}
```

```
void traverse (link ls, void (*fp) (link));

//To call the function
//Function must have matching prototype

traverse(myList,   printNode);
traverse(myList,   printGrade);
```



Generic Types in C

- **Polymorphism**: refers to the ability of the same code to perform the same action on different types of data.

- There are two primary types of polymorphism:

  - Parametric polymorphism: The code takes the type as a parameter, either explicitly (as C++ and Java) or implicitly (say as in C)

  - Subtype polymorphism: Subtype polymorphism is associated with inheritance hierarchies.

- Lectures slides on the topic "Generic Types in C" are drawn from the material available at http://web.eecs.utk.edu/~bvz/cs365/notes/generic-types.html, by Brad Vander Zanden

Generic Types in C

- **Polymorphism in C**:
  C provides pointer to void (for example, void *p), the programmer can create generic data types by declaring values to be of type "void *". For example:

```
struct Node {
      void *value;
      struct Node *next;
};
```

- The programmer can pass in type-specific functions (e.g., comparator functions) that take void *'s as parameters and that downcast the void *'s to the appropriate type before manipulating the data.

- For example, the example on the next page has a generic min function that computes and returns the minimum of two elements. The sample program compares two strings.

```c
#include <stdio.h>
#include <string.h>

// generic min function
void *min(void *element1, void *element2, int (*compare)(void *, void *)) {
    if (compare(element1, element2) < 0)
    return element1;
    else
    return element2;
}

// stringCompare downcasts its void * arguments to char * and then passes
// them to strcmp for comparison
int stringCompare(void *item1, void *item2) {
    return strcmp((char *)item1, (char *)item2);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
    printf("usage: min string1 string2\n");
    return 1;
    }

    // call min to compare the two string arguments and downcast the return
    // value to a char *
    char *minString = (char *)min(argv[1], argv[2], stringCompare);

    printf("min = %s\n", minString);
    return 0;
}
```

Generic Types in C

Advantages

- One copy of the code works with multiple objects.
- The approach supports both generic data structures and generic algorithms.

Disadvantages

- Downcasting can be dangerous, since run-time type checks are not performed in C.
- The code often has a cluttered appearance.

- Lectures slides on the topic “Generic Types in C” are drawn from the material available at http://web.eecs.utk.edu/~bvz/cs365/notes/generic-types.html, by Brad Vander Zanden

Generic Set ADT

- Live Demo of ...
  Generic Set ADT Implementation