Лекция 5. Обзор языка C, продолжение

Управление памятью

При описании языка уже возникали понятия видимости и контекста применительно к переменным и функциям программы. Эти понятия разделяли переменные на две категории по их "времени жизни". Переменные, объявленные внутри блока, возникали в момент входа процесса вычислений в этой блок и уничтожались при выходе из блока. Переменные, объявленные в контексте файла, сохранялись в течение всего времени работы программы и в рамках их области видимости были доступны всегда. Подобное "поведение" переменных обусловлено способом управдения памятью в современных вычислительных системах.

В современных системах оперативная память, в которой работает отдельно взятая задача, в первом приближении можно разделить на два отдельных участка, которые называются heap (куча) и stack (стек).

Куча представляет собой фрагмент памяти, который используется для размещения объектов, которые могут быть доступны в течении всего времени выполнения программы. Например, переменные, объявленные в контексте файлов, размещаются именно там.

Стек используется для размещения параметров функций и переменных, объявленных в блоках этих функций, и некоторой другой служебной информации. При этом это размещение не является постоянным, а создается только в момент вызова функции, а по завершении функции участок стековой памяти освобождается и может быть затем передан другой функции для размещения ее переменных и параметров. Таким образом, переменные внутри блоков функций существуют только во время выполнения функций, а в другой время они не имеют закрепленной за собой памяти.

Механизм работы стека можно проиллюстрировать на таком простом примере

```
int f(int, int);
int g(int);

int main()
{
   int x = 1, y = 2, z;
   z = f (x, y);
   return 0;
}

int f(int a, int b)
{
   int c = g(a) + g(b);
   return c;
}
int g(int s)
```

```
{
   int t = s+2;
   return t;
}
```

Стек: начало:

```
 main:   ret
          x
          y
          z
         ...
    f    ret
         a=x
         b=y
          c
         ...
      g  ret    g  ret
         s=a       s=b
          t         t
         ...       ...
```

Иллюстрация: факториал (как нельзя его вычислять :)))))

```
int Factorial(int k)
{
  if (k==1) return 1;
  return k*Factorial(k-1);
}

int k = Factorial(5);
```

Ну, и как можно :)))

```
int Factorial(int k)
{
  int f = 1;
  while (k) { f *= k--; }
  return f;
}
```

Стек ограничен (хоть и большой).

Переполнение стека при большом количестве локальных переменных и многочисленных вложенных вызовах.

Большие объемы данных нужно размещать в куче.

Объявления в контексте файла. Требование указывать конкретный размер массива — неудачно, если точно не знаем требуемый размер (не хватит или много пропадет зря без использования).

Можно запросить память в куче у системы

```
void * malloc(unsigned int num_bytes);

(size_t --- unsigned int)

sizeof(тип)  или   sizeof(объект)

double *p = (double *)malloc(100*sizeof(double));
if (p==0)    // if (!p)
........
p[i]    i = 0,...,99
........
free(p);
```

выделенная память существует до вызова free, но доступ к ней определяется видимостью соответствующего указателя.

Ошибка: потеря памяти — забыли или не смогли сделать free.

Работа с массивами

Для размещения массивов используем кучу (heap) с запросами malloc и free. Тем самым "экономим"место в стеке и получаем в распоряжение практически всю память, доступную нам в системе.

Удобно сделать свою “библиотеку” для работы с массивами.

```
double * ReadArrayCnt(FILE *f, int *size);    // с подсчетом реального количества
double * ReadArrayNum(FILE *f, int *size);    // по записанному размеру
double * GenerateArray(int size);
void PrintArray(double *a, int n, const char *text);
double SetArray(double *a, int n, double value);
```

Пример: Подсчет min max minmax2.cpp

Комментарии по поводу решения задач с массивами.

Планирование получения и создания массива:

— простейшая инициализация `double a[] = {1., 2., 3.};`

— из файла (с указанием количества или без указания)

— сгенерированный по определенным правилам

Выбор алгоритма:

— трудоемкость $O(N)$ — однопроходный

— трудоемкость $O(N^2)$
— трудоемкость $O(N \log N)$
Планирование процедуры
— параметры — указатель на массив и размер
— специфические параметры алгоритма
— некорректные ситуации (как реагировать)
— — отказ с диагностикой
— — “исправлять” (по умолчаию, по факту, и т.п.)
Тестирование процедуры
— набор данных с характерным распределением
— проверка “просмотром” или формальными условиями